Gerenciamento de Recursos Distribuídos em Sistemas Escaláveis

Jeferson Roberto Marques

Dissertação apresentada ao
Instituto de Matemática e Estatística da
Universidade de São Paulo
para a obtenção do grau de
Mestre em Ciências

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Fabio Kon

São Paulo, fevereiro de 2005.

Gerenciamento de Recursos Distribuídos em Sistemas Escaláveis

São Paulo, fevereiro de 2005.

Comissão julgadora:

- Prof. Dr. Fabio Kon (orientador) — IME - USP
- Prof. Dr. Francisco José da Silva e Silva — DEINF - UFMA
- Prof. Dr. Francisco Reverbel — IME - USP

Agradecimentos

Ao professor Fabio Kon, meu orientador, gostaria de oferecer o primeiro agradecimento. Sua cooperação sempre valorosa em cada etapa da minha pesquisa, sem nunca poupar esforços em ajudar, e sua compreensão foram importantíssimas no decorrer deste trabalho. Aos membros da banca, além do meu orientador, os professores Francisco Reverbel e Francisco Silva e Silva, que expuseram meios de tornar o trabalho mais completo. Meus colegas do Grupo de Sistemas Distribuídos também foram bastante solícitos, promovendo a troca de idéias e a cooperação mútua.

Agradeço imensamente a todos que direta ou indiretamente cooperaram para o desenvolvimento desta pesquisa em toda sua extensão. Os professores do IME, com inestimável competência, oferecem a seus alunos suporte acadêmico de qualidade inquestionável. Somando-se, há os colegas do instituto, que garantem um ambiente agradável para o crescimento intelectual e pessoal de cada um.

Os computadores utilizadas nos experimentos, em sua maioria, foram gentilmente cedidos pelo professor Francisco Ballesteros do Grupo de Sistemas y Comunicaciones da Universidade Rey Juan Carlos, sediada na Espanha. Fico muito grato ao professor por ter concedido acesso às máquinas.

À minha família, certamente, sempre presente nos momentos mais preciosos, fico imensamente grato.

Sumário

1 Introdução **6**
1.1 Gerenciamento de recursos com apoio a QoS 8
1.2 Aplicações distribuídas sensíveis a QoS 9
1.3 Organização da Dissertação 10

2 2K – Um sistema operacional distribuído para ambientes heterogêneos e dinâmicos **12**
2.1 Gerenciamento de recursos 13
2.1.1 Arquitetura e implementação 14
2.1.2 Arquitetura intra-aglomerado 15
2.1.3 Disseminação das informações 16
2.1.4 Tolerância a falhas (LRM e GRM) 16
2.1.5 Descoberta, iniciação e execução de serviços fundamentais 17
2.1.6 Serviços e operações fundamentais 17
2.1.7 Carga e execução de componentes 18

3 Extensão do modelo **21**
3.1 Arquitetura inter-aglomerado 22
3.2 Disseminação de informações 23
3.2.1 Descrição do protocolo 24
3.2.2 Implementação 26
3.3 Localização de recursos e execução de componentes 28
3.3.1 Descrição do protocolo 28
3.3.2 Métodos para escolha de candidatos 29
3.3.3 O protocolo 29
3.4 Iniciação dos serviços 34
3.5 Funcionalidades não implementadas 35

4 Análise de desempenho **37**
4.1 Uma abordagem sistemática para avaliação de desempenho 37
4.2 Aplicação dos passos 38
4.2.1 Definição dos objetivos e descrição do sistema 38
4.2.2 Descrição dos serviços e dos propósitos 38
4.2.3 Seleção das métricas 39
4.2.4 Seleção dos parâmetros 40
4.2.5 Seleção dos fatores a serem estudados 40
4.2.6 Seleção da técnica de avaliação 41
4.2.7 Seleção da carga a ser aplicada 41

4.2.8 Elaboração dos experimentos . . . 41
4.3 Coleta das métricas . . . 45
4.4 Sistema de coleta de informações de desempenho . . . 46
4.5 Sistema para geração de carga . . . 48
4.5.1 Carga para consumo de processamento . . . 48
4.5.2 Automatização do processo . . . 49
4.6 Análise, apresentação e interpretação dos dados . . . 50
4.6.1 Sistema de disseminação de informações . . . 50
4.7 Conclusões dos Experimentos . . . 57

5 Trabalhos Relacionados **59**
5.1 Globus . . . 59
5.2 Condor-G . . . 61
5.3 Legion . . . 62
5.4 GriT . . . 63

6 Conclusão **65**
6.1 Trabalhos futuros (complementações) . . . 65
6.1.1 Alternativa de comunicação inter-aglomerados . . . 65
6.1.2 Segurança . . . 66
6.1.3 Inclusão de recursos . . . 67
6.1.4 Diagnóstico e gerenciamento de rede . . . 67
6.1.5 Tolerância a falhas . . . 67

A Equações **73**

B Consumo artificial de processamento **74**

C Simulação de solicitação de execução de componentes **76**

D Especificação das máquinas usadas nos experimentos **77**

Resumo

O desenvolvimento de redes de computadores nas últimas décadas trouxe consigo um aumento da complexidade das aplicações distribuídas. Tal nível de sofisticação tornou possível o acesso a uma gama variada de recursos computacionais.

O trabalho apresentado nesta tese de Mestrado baseia-se no sistema operacional 2K, o qual possibilita a interação entre componentes em arquiteturas heterogêneas e a adaptação dinâmica a variações ocorridas no ambiente de execução. O Sistema de Gerenciamento de Recursos do 2K foi estendido de forma a incrementar a escalabilidade através da integração de coleções de aglomerados geograficamente distantes em uma grade única do 2K.

O sistema, após implementado, passou por uma análise sistemática para avaliação de seu desempenho, constituída de uma série de experimentos onde foram observadas métricas que representavam seu comportamento, como consumo de memória e processamento. O objetivo dessa análise foi mensurar, em termos qualitativos e quantitativos, o impacto do sistema no ambiente onde é executado.

Outros sistemas, funcionalmente semelhantes, foram investigados e tiveram suas características comparadas com as do sistema apresentado nesta dissertação.

Abstract

The development of computer networks in the last decades led to an increase in the complexity of distributed applications. This level of sophistication made possible the access to a wide variety of computational resources.

The work presented in this Master thesis is based on the 2K operating system, which supports interactions among components in heterogeneous architectures and dynamic adaptation to oscillations in the execution environment. The 2K Resource Management System was extended to improve scalability through aggregation of a large collection of geographically distributed clusters in a single 2K grid.

After deployed, a systematic analysis was applied to the system to evaluate its performance. The analysis was provided by the observation of metrics that represented its behavior, such as memory consumption and processor usage. The objective of the analysis was to measure the impact of the system on the execution environment, in qualitative and quantitative forms.

Other functionally similar systems were studied and compared to the system presented in this thesis.

Capítulo 1

Introdução

O desenvolvimento crescente de tecnologias direcionadas para infra-estrutura de redes de comunicação de computadores trouxe consigo a possibilidade de surgirem novas aplicações cada vez mais complexas e destinadas a variados fins. Antigas concepções, antes impossibilitadas de serem praticadas pela falta de ambientes propícios, podem ser finalmente implementadas. Aliados a esses avanços, os sistemas atuais são capazes de lidar, por exemplo, com dados multimídia de diversas categorias — texto, som, imagens — e com computações intensivas de grande escala — engenharia, bancos de dados, meteorologia. Desenvolvedores e usuários já usufruem dos benefícios concretizados por meio destes avanços, como o aumento da produtividade decorrente da facilidade de acesso e usabilidade de suas aplicações.

No entanto, a concretização destes sistemas eventualmente depara-se com obstáculos relativos a custos e complexidades inerentes à infra-estrutura sobre a qual as aplicações devem ser executadas. Sistemas que exigem alta quantidade de recursos, como ambientes colaborativos avançados, que são sustentados pela determinação da qualidade de serviço, são exemplos que necessitam destas tecnologias e dos ambientes delas decorrentes.

Podemos definir qualidade de serviço como uma especificação de recursos computacionais que sejam mensuráveis e que, possivelmente, o sistema responsável forneça garantia sobre a reserva contida nessas especificações, inclusive previamente. Dessa maneira, um usuário — ou aplicação, equivalentemente — que necessite de uma determinada quantidade de memória mínima para poder executar um programa, terá seu pedido atendido integralmente, ou terá seu pedido rejeitado. É preciso identificar as relações entre as necessidades do usuário e os recursos computacionais de forma que este mapeamento reflita o desejo do usuário ou aplicação.

Um sistema de gerenciamento de recursos deve oferecer apropriadamente qualidade de serviço para cada aplicação, baseando-se nas características da aplicação, nos recursos disponíveis e no estado do sistema. É desejável, portanto, que existam políticas e mecanismos para garantir que as aplicações obtenham informações dinâmica e periodicamente sobre a disponibilidade dos recursos de um sistema.

O desempenho de sistemas baseados em rede é consideravelmente dependente da latência e da largura de banda disponíveis. Assim, torna-se desejável que as aplicações sejam adaptáveis e sensíveis a oscilações nestes ambientes, exigindo que valores de determinadas métricas estejam dentro de intervalos pré-estabelecidos e tolerando eventuais indisponibilidades do recurso desejado.

Num ambiente de aglomerado (*cluster*), uma aplicação obtém a vantagem de usufruir de diversos recursos isoladamente ou em conjunto, a fim de atingir seus propósitos em adição ao usualmente disponível numa única máquina. Estes ambientes permitem que uma aplicação

utilize-se de poder computacional que não teria a possibilidade de obter isoladamente, isto é, nos limites da própria máquina onde é executada.

A respeito da topologia em que é estruturado, um aglomerado normalmente é construído dentro de uma rede local, favorecendo-se da velocidade de transmissão de dados e permitindo um gerenciamento simples conseqüente da própria estrutura.

No entanto, tais aglomerados são limitados enquanto mantêm seus recursos isolados dentro de uma rede local com acesso restrito a usuários autorizados e submetidos às políticas geridas pelos administradores. De certa forma, este modelo protege os dados e os recursos de invasores externos. Porém, mesmo que o administrador permita que os recursos computacionais que gerencia sejam oferecidos para um público extra, além dos usuários autorizados em seus domínios, os mecanismos usados nestes aglomerados não fornecem um meio simples e seguro para que usuários externos utilizem as capacidades computacionais presentes em seu domínio. Outro ponto limitante deste modelo é a dificuldade, senão impossibilidade, de conectar dois ou mais aglomerados diferentes de forma que seus participantes passem a ter acesso aos recursos remotos transparentemente.

Para contornar o problema acima descrito, um novo modelo foi desenvolvido. Valendo-se da implantação e da evolução de tecnologias recentes de redes de alta velocidade, aliados à proliferação de máquinas de baixo custo e alto desempenho, surgiram os ambientes de meta-computação (*networked virtual computing* [FK97]). Estes ambientes podem combinar, potencialmente, centenas de máquinas e domínios administrativos conectados por uma rede local ou eventualmente global, como a Internet. São as chamadas grades computacionais (*computational grids* [CFK+98; FKNT02; Fos01; FK98]), infra-estruturas de *middleware* que oferecem acesso uniforme e gerenciamento para recursos computacionais possivelmente distribuído por várias localidades e diferentes domínios, ou seja, muitos pontos são encapsulados e ocultados a fim de garantir que a heterogeneidade do ambiente torne-se imperceptível para o usuário, além de assegurar outras características próprias de sistemas distribuídos, como tolerância a falhas, disponibilidade e concorrência [CC91], ainda que não sejam orientados a objetos.

Usualmente, as aplicações são desenvolvidas para uma única arquitetura e em ambientes com comportamento consideravelmente previsível. Contrastando com este cenário, aplicações para grades computacionais potencialmente necessitam de recursos variados que são requisitados e alocados dinamicamente de acordo com a disponibilidade. Políticas e mecanismos para coordenação eficiente destes recursos são, portanto, cruciais para o funcionamento de uma grade computacional.

O propósito do ambiente de grade é suportar compartilhamento de recursos entre conjuntos dinâmicos de indivíduos e instituições, ou seja, numa visão pragmática, compartilhamento expandido para além da rede local. Este ambiente deve permitir acesso consistente e de baixo custo aos recursos independentemente de sua posição física, além de oferecer uma infra-estrutura que suporte sucessivas expansões no sistema, isto é, alto grau de escalabilidade. Entre as aplicações e recursos que as grades podem oferecer estão: compartilhamento de instrumentos científicos avançados, tele-imersão, computação intensiva e supercomputação distribuída.

Nesse contexto, uma grade computacional deve suportar garantia quanto a restrições sob disponibilidade de recursos, as quais, não raramente, representam um conjunto complexo de itens, além das restrições relativas a disponibilidade: segurança, tolerância a falhas, tempo de resposta e vazão.

Um importante fator a ser considerado também é a autonomia de cada aglomerado de acordo com a política adotada para suas dependências, que deve ser mantida conforme os critérios de quem o gerencia. Dessa forma, a liberdade de controle que possui cada grupo deve ser mantida.

Os modelos recentes de grades e de seus sistemas correlatos abordam soluções hipotéticas para diversas dificuldades nos sistemas computacionais atuais, além de promoverem um incremento na eficiência das aplicações. No entanto, uma extensa pesquisa encontra-se em andamento em várias subáreas pertinentes, pois uma grade computacional não é simplesmente um conjunto variado de serviços integrados. Uma gama significativa de problemas ainda deve ser solucionada mesmo nos projetos mais recentes para atingir a proposta essencial deste tema.

O projeto desenvolvido e apresentado nesta dissertação visa estender determinadas funcionalidades do 2K, um sistema operacional que suporta interação entre componentes pertencentes a arquiteturas heterogêneas, além de adaptar-se dinamicamente às variações sofridas pelo ambiente onde é executado [Kon00; KCM+00]. De acordo com Ian Foster, num artigo [Fos02] em que discorre sobre a definição de grade computacional, um sistema pode ser assim classificado se passar pelo crivo de 3 critérios essenciais. Assim, o sistema deve:

1. coordenar os recursos disponíveis de modo que diminua a probabilidade de escassez;

2. ter sido desenvolvido sobre o alicerce de protocolos e interfaces que sejam padrões e de múltiplas finalidades, como autenticação e descoberta de recursos. Os elementos padronizados permitem, por exemplo, estabelecer dinamicamente configurações diferentes de recursos compartilhados entre quaisquer participantes, tornando essas configurações compatíveis.;

3. oferecer suporte a qualidade de serviço complexa, como a combinação de restrições feitas por uma aplicação com respeito a diferentes recursos.

O 2K, no entanto, descumpre o especificado no segundo critério de Foster por não possuir protocolos padronizados. Ainda que tenha sido desenvolvido com o padrão CORBA, existem protocolos subjacentes ao funcionamento do 2K que são específicos deste sistema.

Em específico, será estendido o Sistema de Gerenciamento de Recursos [Yam00] deste sistema operacional de maneira a permitir que um maior número e variedade de recursos sejam incorporados e compartilhados por todos os participantes do sistema, nos limites de um aglomerado e entre aglomerados.

1.1 Gerenciamento de recursos com apoio a QoS

O sistema responsável pelo gerenciamento de recursos é um componente de considerável importância num ambiente computacional distribuído. A capacidade de oferecer suporte a qualidade de serviço — *Quality of Service* (QoS) — por parte do sistema de gerenciamento de recursos permite um ajuste mais fino e grau maior de justiça quanto a sua utilização pouco previsível pelas aplicações, restringindo a interferência potencialmente negativa que acarreta o uso concorrente de um recurso específico. Além disso, mantendo um sistema que restrinja e gerencie os recursos de acordo com a demanda, é possível evitar o uso indevido e, assim, minimizar a probabilidade de que o pedido de uma aplicação não seja atendido. Como exemplo de recursos ou serviços que possuem sua qualidade de serviço gerenciado pode-se incluir segurança, coordenação de diversos recursos — como forma de atender restrições complexas agregando várias métricas, tempo de resposta, memória, período de uso de processamento e garantia de determinado nível de vazão pela rede.

Tal sistema deve oferecer apropriadamente qualidade de serviço para cada aplicação, segundo suas exigências pré-estabelecidas, baseando-se nas características da aplicação, nos recursos disponíveis e no estado do sistema. É desejável, portanto, que existam políticas e mecanismos

para garantir que o sistema ou as aplicações obtenham informações dinâmica e periodicamente sobre a disponibilidade dos recursos, a fim de que possa ser devidamente elaborada uma configuração mais coerente com o cenário descrito pelos dados obtidos sobre o estado dos recursos disponíveis.

Um sistema de gerenciamento de recursos com apoio a QoS pode ser inserido, por exemplo, no sistema operacional ou pode constituir parte integrante de um *middleware*. Este pode oferecer o suporte, adaptando-se de acordo com as condições e prioridades impostas pela aplicação face aos recursos, ou ainda permitindo que esta adaptação seja delegada para as aplicações, de forma que estas sejam responsáveis por adaptar-se individualmente.

Inserir essas características no *middleware* elimina a onerosa tarefa de introduzir mudanças no sistema operacional, além de não interferir nas suas políticas e propriedades originais já extensamente avaliadas pelos desenvolvedores e usuários, como a implementação de comunicação em rede. Dessa forma, transparentemente, a camada intermediária entre o sistema operacional e aplicação adquire a responsabilidade de gerenciar os recursos adequadamente para que não haja escassez em alguns participantes do sistema, enquanto que em outros exista oferta abundante. Por meio de políticas e mecanismos apropriadamente concebidos para esse fim, é possível atingir um equilíbrio aceitável quanto às ofertas de recursos, assim como o nível disponível de cada um.

Outra característica de um sistema com apoio a QoS refere-se à possibilidade de garantir a disponibilidade dos recursos solicitados por um usuário no momento em que a requisição é analisada e pode ser atendida. Isso é concretizado por meio de reserva dos recursos de antemão. Com a requisição aceita e reforçada pela reserva dos recursos nela descritos, impede-se que o usuário experimente surpresas indesejadas no momento em que sua aplicação será executada.

1.2 Aplicações distribuídas sensíveis a QoS

Aplicações distribuídas podem ser favorecidas pelas vantagens de um sistema que oferece suporte a qualidade de serviço. Combinando o caráter distribuído de uma aplicação que faz uso de diferentes recursos computacionais e a garantia prévia de disponibilidade destes recursos proporcionada pelo suporte a qualidade de serviço, é possível, por exemplo, atingir um grau pré-determinado de justiça no compartilhamento dos recursos disponíveis, melhor gerenciamento do sistema e imposição de prioridade no atendimento a aplicações, se assim desejado.

De acordo com as exigências que devem ser atendidas com respeito a uma variedade de recursos, entre outras características intrínsecas, aplicações multimídia distinguem-se visivelmente de outras aplicações que fazem uso de qualidade de serviço para alcançar o funcionamento adequado. Vídeo-conferência é uma categoria de aplicação multimídia que demanda, por exemplo, um intervalo restrito de vazão de dados na rede entre os usuários desse serviço. Para obter-se uma apresentação de qualidade satisfatória, confortável para o usuário assistir e ouvir, os dados relativos ao áudio e ao vídeo devem estar submetidos a determinados valores mínimos de recursos, além de manterem uma certa continuidade dos dados para que sejam processados no período apropriado.

Em ambientes inerentemente heterogêneos, como em grades computacionais, é necessário uma oferta equilibrada em razão da demanda por parte das aplicações. Dessa forma, hipoteticamente, uma aplicação que necessite como recursos essenciais para sua execução o uso simultâneo e pré-definido de espaço para armazenamento, processamento intensivo e vazão de dados pela rede, deve ser atendida por uma infra-estrutura que esteja apta a reservar a alocar tais recursos. Tais características podem ser atingidas com base em mecanismos e políticas executados

por subsistemas objetivando a busca e seleção dos recursos apropriados obedecendo sempre as restrições da aplicação.

1.3 Organização da Dissertação

Esta dissertação é constituída de um capítulo introdutório no qual o modelo de sistema distribuído aqui tratado é particularizado e o objetivo do projeto é exposto. A apresentação inicial converge para o gerenciamento de recursos com apoio a qualidade de serviço (QoS) em aplicações distribuídas e, especialmente, no 2K.

O Capítulo 2 apresenta o 2K, um sistema operacional distribuído para ambientes heterogêneos e dinâmicos. Sua arquitetura e implementação são discutidas sem estender-se em detalhes. Por outro lado, seu sistema de gerenciamento de recursos, onde se baseia a maior parte desta dissertação, é introduzido e pormenorizado. A arquitetura intra-aglomerado do sistema de gerenciamento de recursos é caracterizada juntamente com o mecanismo de disseminação de informações sobre recursos dentro de um aglomerado, assim como a localização e execução de componentes. Em seguida, para embasamento técnico do leitor acerca do sistema, os serviços e operações fundamentais são demonstrados e o capítulo é finalizado com o detalhamento do método utilizado para carga e execução de componentes nos limites de um aglomerado.

O próximo tópico, presente no Capítulo 3, envolve a extensão do modelo apresentado no Capítulo 2, enfatizando o centro desta expansão de funcionalidades: a arquitetura inter-aglomerado. A seguir, é apresentada em detalhes a política e o mecanismo que sustentam a disseminação de informações entre aglomerados, base do sistema, pois ele é responsável por manter os nós atualizados sobre o estado dos recursos em cada nó de uma maneira abrangente, por meio de indicações aproximadas de onde os recursos solicitados podem ser encontrados. O protocolo é exposto em detalhes e o algoritmo da sua implementação comentado passo a passo. Discutido este subsistema, segue a exposição do subsistema de localização de recursos e execução de componentes. De modo similar ao subsistema de disseminação de informações, sua apresentação consiste na explicação do protocolo, dos passos envolvidos no algoritmo e da ilustração de um caso particular do protocolo em ação, desde a requisição do usuário até a descoberta do local apropriado para obtenção dos recursos solicitados.

A fim de obter dados representativos sobre o comportamento do sistema, foram elaborados procedimentos sistemáticos para coletar métricas relevantes e para interpretar as informações obtidas. No Capítulo 4 é discutida a metodologia aplicada nesse processo baseado em experimentos. Inicialmente, são definidos os objetivos da análise de desempenho e descritas funcionalidades do sistema. As métricas são escolhidas para quantificar o desempenho e os recursos variáveis (fatores) são delimitados. Dentre os parâmetros que geram impacto significativo no sistema, um conjunto é separado para estudo detalhado. A carga que deve ser aplicada em alguns recursos para causar distúrbios em seus estados é definida em seguida. A fim de conhecer os efeitos dos parâmetros selecionados, os experimentos, então, são definidos e caracterizados. A coleta das informações sobre o estado do sistema durante seu funcionamento durante os experimentos foi tarefa de um programa com este propósito, exclusivamente, cujo modelo e arquitetura são apresentados. Expostos os elementos essenciais para a execução dos experimentos, exibe-se a automatização de todo o processo, o qual envolve uma quantidade significativa de máquinas. Finalizando o capítulo, as informações coletadas são interpretadas com base em dados estatísticos e são apresentados gráficos contendo os dados coletados.

Os trabalhos relacionados a este projeto são discutidos no Capítulo 5. O primeiro sistema apresentado é o Globus e, em especial, seus serviços de alocação de recursos e de descoberta de

serviços. O Condor-G, estreitamente relacionado ao Globus aparece após o Globus. O Legion, um sistema de grade implementado com base em orientação a objetos vem em seguida. O Nimrod/G assemelha-se ao Condor-G e também destina-se a gerenciar e escalonar computações acerca de recursos dinâmicos. Por último são expostas as características do GriT, também baseado no TAO, assim como o trabalho exposto nesta dissertação. Ele faz uso do CORBA Component Model [Gro00a] (CCM).

No Capítulo 6, destinado à conclusão, na Seção 6.1 são discutidas alguns pontos não implementados neste projeto, mas que possuem importância relevante para gerenciamento de recursos em sistemas distribuídos em geral. Primeiramente, é discutida uma alternativa à comunicação entre aglomerados em que se baseia esta dissertação. Após este tema, a questão da segurança é abordada, como a interceptação de dados na comunicação entre participantes e autorização de uso sobre os recursos disponíveis em um aglomerado. Em seguida, apresenta-se uma opção para aumentar o dinamismo da inclusão de novos recursos ao sistema. Finalizando, discute-se medidas que podem auxiliar a diminuir a suscetibilidade do sistema a eventuais falhas.

Capítulo 2

2K – Um sistema operacional distribuído para ambientes heterogêneos e dinâmicos

O 2K [KCM+00] é um sistema operacional distribuído desenvolvido na Universidade de Illinois. Ele incorpora funcionalidades que permitem interação entre componentes[1] implantados em sistemas com arquiteturas potencialmente heterogêneas, capacidade de adaptar-se dinamicamente em tempo de execução e gerenciamento de recursos e das dependências entre componentes.

O dinamismo da rede e da disponibilidade dos recursos são pontos decisivos para o desenvolvimento de um sistema como o 2K. É factível que mudanças radicais ocorram nos elementos do ambiente. Quanto mais suscetível a alterações estiver o ambiente, maior a complexidade no desenho da estrutura base, o que implica em desafios ainda maiores na elaboração de políticas e mecanismos sejam eles responsáveis pelo gerenciamento dos recursos ou das dependências.

O sistema desenvolvido é inteiramente orientado a objetos e os elementos referentes a recursos de hardware e software são representados como objetos CORBA. Os serviços oferecidos pelo sistema, como o de nomes e de negociação, são serviços CORBA. Dessa forma, uma parcela da heterogeneidade intrínseca ao sistema é eliminada pela visão de uniformidade garantida pelo CORBA.

Cada entidade possui uma identidade global e uma lista que indica de quais outras entidades ela é dependente. Quando um usuário utiliza um objeto qualquer, o sistema encarrega-se de executar apropriadamente os objetos correlacionados de acordo com seus requisitos previamente especificados.

O 2K foi implementado usando o dynamicTAO [KRL+00], extensão de uma implementação da arquitetura CORBA chamada TAO, mais especificamente o ORB TAO [SGH+97], que se vale de padrões para concorrência e sistemas distribuídos [SSRB00]. O dynamicTAO é um ORB reflexivo, ou seja, ele permite reconfigurações dinâmicas, em tempo de execução, tanto de suas próprias características internas como das características de aplicações que o utilizam e sem a necessidade de reiniciar o ORB. Favorecendo-se dos serviços CORBA já existentes como os Serviços de Nomes, de Negociação, de Segurança e de Persistência, estendeu-se o modelo CORBA incrementando-o com outros serviços quase imprescindíveis num sistema com o objetivo como o do 2K. Os serviços em questão são: Serviço de Gerenciamento de Recursos, Configuração Automática e Distribuição de Código [KCM+00], ilustrados na Figura 2.1.

[1] O termo "componente", nesta dissertação, corresponde tanto a uma aplicação completa como a um elemento integrante de uma aplicação distribuída.

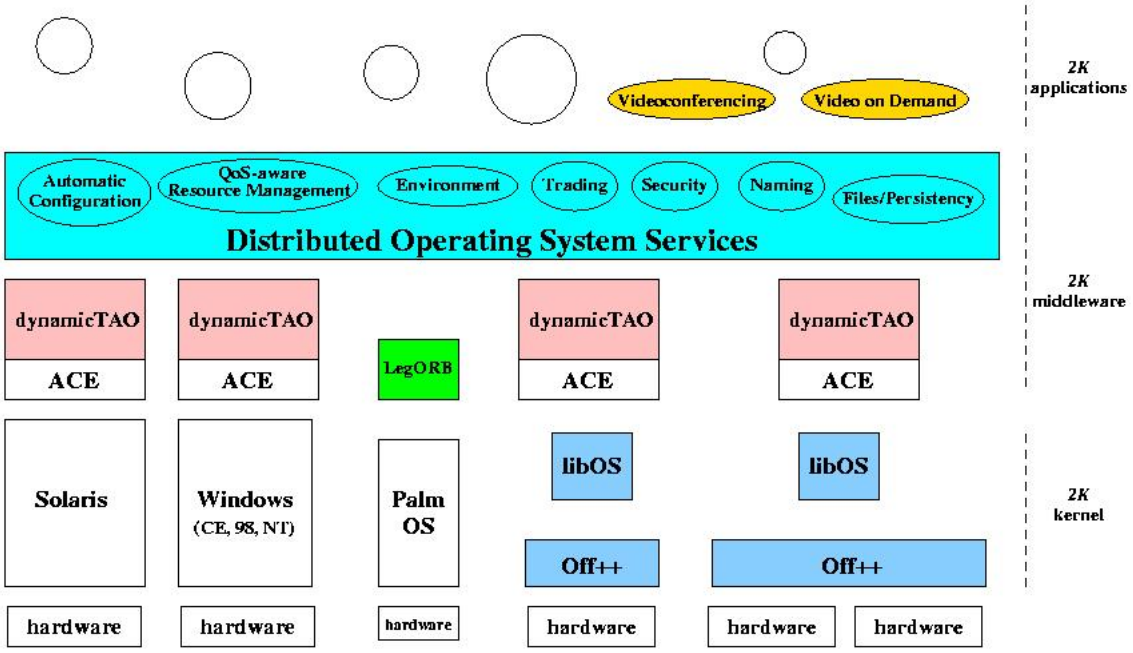

Figura 2.1: *Arquitetura do 2K*

2.1 Gerenciamento de recursos

O 2K [KCM+00], sendo um sistema distribuído, apresenta diversos obstáculos que devem ser transpostos para obter um sistema de gerenciamento de recursos compatível com a finalidade deste sistema operacional, assim como em outros sistemas distribuídos com características semelhantes.

Vários aspectos devem ser analisados ao desenvolver um sistema de gerenciamento de recursos. Em particular, é preciso considerar características como reserva de recursos para as aplicações antes de iniciar o procedimento para execução, disseminação de informações relevantes, procura de recursos disponíveis e acompanhamento do estado dos recursos da máquina.

A disseminação de informações é um ponto crucial para o sistema e necessita de atenção especial. Uma qualidade peculiar a sistemas distribuídos é a volatilidade das informações que representam o estado de um determinado recurso; quantidade de memória livre, carga no processador e espaço em disco são alguns recursos essenciais que podem apresentar mudanças significativas dentro de um curto período de tempo. Dessa forma, as políticas que regem o controle do fluxo dos dados coletados devem ser cuidadosamente elaboradas para que a procura e a atualização dos dados sejam eficientes o bastante. É necessário considerar a freqüência com que os repositórios de dados são atualizados, ou seja, se eles forem atualizados com uma freqüência elevada, pode ocorrer um impacto negativo no sistema, ainda que eles reflitam o estado dos recursos com maior precisão.

O suporte a qualidade de serviço (QoS) constitui outro item de importância significativa no 2K. Ele foi desenvolvido para oferecer meios de estabelecer restrições impostas por uma aplicação ou usuário, isto é, a aplicação pode definir previamente quais os recursos mínimos necessários para sua execução, assim como os limites desejados. O sistema de gerenciamento de recursos oferece mecanismos para garantir que estas exigências sejam cumpridas.

Comumente, uma aplicação reserva recursos através de bibliotecas que permitem realizar esta ação somente na própria máquina, localmente. Como conseqüência, surge uma situação indesejada: se estas bibliotecas forem simplesmente acionadas por intermédio de objetos CORBA, os recursos serão reservados para o servidor e não para o cliente que fez efetivamente a requisição. No 2K, este problema é contornado de forma a permitir o uso destas bibliotecas em ambientes distribuídos, como será descrito posteriormente.

O componente central do sistema de gerenciamento de recursos no 2K é o Serviço de Negociação (*trader*) CORBA. No 2K, particularmente, ele é responsável por armazenar as informações sobre o estado dos recursos disponíveis nas máquinas que constituem o aglomerado. A escolha pelo *trader* traz vários benefícios, principalmente em relação às funcionalidades oferecidas e robustez. Diferentemente de outros sistemas para reserva de recursos, não foi desenvolvida uma linguagem nova para especificar consultas por recursos; a própria solução projetada pela OMG para o *trader* foi utilizada: a *Trader Constraint Language* [HV99].

2.1.1 Arquitetura e implementação

O Serviço de Gerenciamento de Recursos foi desenvolvido sobre o middleware dynamicTAO, que é baseado em uma implementação do CORBA chamada TAO. O dynamicTAO acrescenta funcionalidades ao ORB TAO para permitir, essencialmente, reconfigurações em tempo de execução. O autor deste serviço foi Tomonori Yamane e o trabalho faz parte de sua dissertação de mestrado [Yam00].

O TAO é uma implementação do ORB padronizado pela OMG implementado em C++, baseado no arcabouço ACE [Sch], o qual possui padrões de projeto para comunicação concorrente, como base de sua arquitetura. As áreas de atuação para a qual foi conceitualmente projetado incluem sistemas de tempo-real, oferecer novos padrões direcionados para sistemas distribuídos e suporte para aplicações que impõem determinada qualidade de serviço para serem executadas.

As informações no *trader* são armazenadas na forma de pares $(nome, valor)$, com *nome* correspondendo a um determinado recurso e *valor* a uma quantidade disponível ou, se aplicável, total do recurso. Os serviços a serem oferecidos (exportados) devem ser registrados juntamente com suas propriedades e seus respectivos valores. No entanto, é possível que o valor de uma propriedade varie e, assim, não ser possível obter seu valor no momento do registro. Propriedades que se enquadram nessa categoria são denominadas propriedades dinâmicas; ao registrar tais propriedades, deixa-se indicado sua natureza, juntamente com uma referência para o objeto representante da mesma. Dessa forma, toda vez que uma consulta for feita ao *trader* relativa a esta propriedade, o *trader* encarrega-se de fazer uma chamada para objeto, repassando a consulta, o qual responde informando o valor da propriedade naquele momento.

Um *trader* participa de um aglomerado formado por uma ou mais máquinas, sem restrições quanto à localidade. Assim, ele fica responsável pelas informações referentes ao estado de cada recurso disponibilizado por cada uma das máquinas. Ele pode, inclusive, estar numa das máquinas que oferecem recursos, sem grandes impactos na carga do processamento e sem consumir muita memória, tomando-se como parâmetro a configuração básica que usualmente encontramos em computadores atualmente.

A reserva de recursos é executada pelo *CPU Server* [NhCN99], uma vez que não existe na especificação do *Trader CORBA* funcionalidade análoga. O *CPU Server* é um escalonador com suporte a aplicações de tempo-real capaz de agendar tarefas periódicas e de antever a viabilidade de executar uma aplicação, entre outras funções.

2.1.2 Arquitetura intra-aglomerado

Três componentes centrais constituem o sistema de gerenciamento de recursos do 2K: o Gerenciador de Recursos Global (*Global Resource Manager* - GRM), o Gerenciador de Recursos Local (*Local Resorce Manager* - LRM) e o Repositório de Componentes (*Component Repository* - CR), com a possibilidade de haver mais de um LRM ou CR no aglomerado em funcionamento simultaneamente, como ilustrado na Figura 2.2.

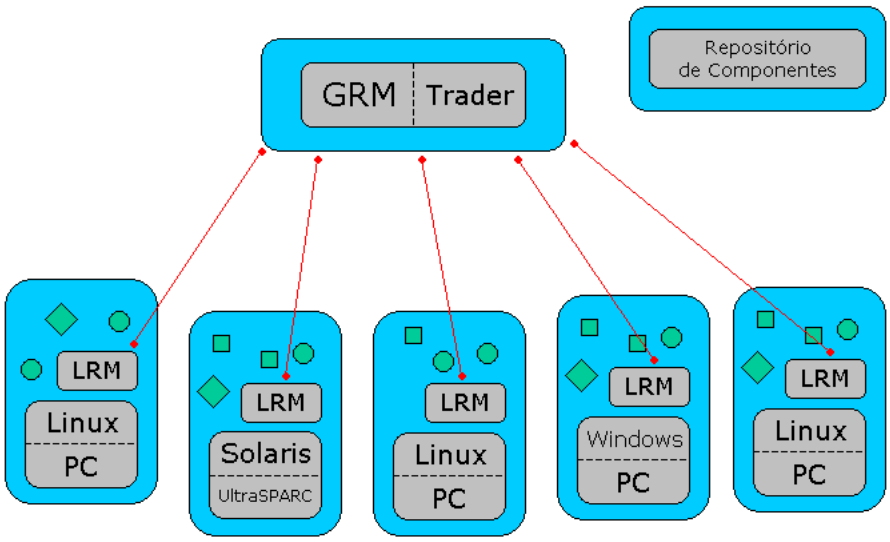

Figura 2.2: *Possível organização de um aglomerado*

Pela arquitetura e política concebidas, é permitido que exista somente um GRM operando num aglomerado, em razão de sua própria finalidade: gerenciar informações sobre LRMs. O GRM é essencialmente uma extensão do *Trader* CORBA, ao qual foram incorporadas funções como o monitoramento dos LRMs nele registrados, a fim de diagnosticar eventuais quedas de algum participante. Adicionalmente, ele desempenha outra função importante no sistema: encaminhar apropriadamente as requisições que nele chegam pelos clientes, os LRMs, para outros LRMs.

Em cada uma das máquinas participantes no aglomerado deve existir obrigatoriamente um único LRM. O motivo é análogo ao do número de GRMs necessários num aglomerado; como o LRM combina em si todos os processos envolvidos para atender a requisições de aplicações e trabalha como intermediador entre as aplicações e outras máquinas, é natural que a relação entre LRM e máquina seja de um para um, isto é, um LRM em cada máquina do aglomerado. O LRM, em cooperação com alguns outros componentes, coordena o processo de criação e execução de tarefas requisitadas diretamente por um cliente. Este processo envolve encontrar o objeto requisitado que representa a tarefa a ser executada, reservar recursos segundo as restrições impostas pelo cliente, iniciar a execução e retornar uma referência para o objeto (componente).

O repositório de componentes (CR) é um integrante auxiliar do sistema que armazenam e disponibiliza componentes que podem ser utilizados por todo o aglomerado, além de manter informações sobre os requisitos e dependências do componente. Ele se baseia no Repositório de Componentes (*Component Repository*) do dynamicTAO [KRL+00].

2.1.3 Disseminação das informações

O subsistema responsável por difundir informações sobre cada máquina é um elemento especial no contexto do sistema completo. É preciso obter informações em períodos satisfatórios e transmiti-las de forma razoável para não prejudicar o desempenho do sistema, seja por inserir informações em excesso na rede — por vezes desnecessárias — ou por falta destas, resultando na descrição imprecisa do cenário, isto é, da disponibilidade dos recursos.

A propagação das informações que representam o estado dos recursos está vinculada diretamente ao modo como os recursos são monitorados e à política de atualização adotada em cada LRM, como exposto a seguir.

O monitoramento dos recursos locais é feito periodicamente pelo LRM, que se utiliza das funcionalidades oferecidas pelo DSRT [NhCN99], que representa o elo, um intermediário com o sistema operacional. O DSRT (*Dynamic Soft Real-Time*) é uma biblioteca que provê operações que compreendem responsabilidades como: reserva de recursos da máquina, como processamento e memória, informação do grau de disponibilidade dos recursos e, como seu próprio nome diz, escalonamento de processos baseados em tempo-real flexível, isto é, com uma margem permitida de diferença se o tempo determinado não puder ser satisfeito. O DSRT foi utilizado como forma de suprir uma limitação natural do *trader*: a impossibilidade de efetuar reserva de recursos.

De acordo com o dos dados obtidos pela consulta, o GRM é informado, se for necessário, das mudanças decorridas desde a última utilização. Como conseqüência desta política, evita-se o uso desnecessário da rede, diminuindo o tráfego destas informações. A política adotada implica em duas situações nas quais o LRM executa uma requisição com o propósito de atualizar a base de dados do GRM responsável:

1. A utilização do recurso sofreu uma mudança significativa dentro de um intervalo predeterminado. Cada LRM verifica periodicamente o estado dos recursos locais, como memória física disponível e carga no processador. Se houver uma alteração considerada significativa em relação ao estado anterior, como uma variação de 10%, por exemplo, é enviada uma notificação para que a base de dados do GRM seja atualizado;

2. Um período limite sem atualizações expirou. Este mecanismo é utilizado com o propósito de manter o conjunto dos participantes registrados do sistema atualizado, ou seja, a requisição com a atualização tem o efeito de um *keep-alive*. Assim, a inacessibilidade de uma máquina é detectada pela ausência desta requisição. Outra finalidade, porém com menor importância, é manter as informações da base de dados mais próximas do valor real dos recursos nas máquinas.

2.1.4 Tolerância a falhas (LRM e GRM)

Eventualmente, um LRM pode tornar-se indisponível devido a inúmeras razões, como indisponibilidade de acesso à rede, interrupção inadvertida do LRM — o processo — da máquina, dano físico em algum componente da máquina, entre outras. Tal circunstância foi prevista e é tratada pelo sistema para que este continue operando adequadamente, sem comprometer o atendimento às requisições dos demais LRMs, na medida do possível.

O GRM é capaz de identificar essas condições de inoperância por ocasião dos seguintes indícios:

- uma requisição não pode ser encaminhada porque o objeto remoto encontra-se inacessível;

- o período máximo configurado para o LRM efetuar a atualização dos recursos no GRM expirou.

Nos dois casos presume-se que a máquina onde localiza-se o LRM não está adequadamente operante ou há uma eventual instabilidade na comunicação pela rede. Em resumo, encontra-se inatingível e, por conseguinte, o LRM é retirado da base de dados do GRM, não fazendo mais parte do aglomerado.

Por razões similares às que resultam indisponibilidade do LRM, é possível também que o GRM não responda a tentativas de contato. Nesse caso, o procedimento adotado para a falha é o LRM encontrar outro GRM para registrar-se e dar continuidade a sua tarefa de encaminhar requisições dos clientes.

2.1.5 Descoberta, iniciação e execução de serviços fundamentais

Primeiramente, antes de descrever os mecanismos de como é feita a descoberta de serviços pelos clientes e pelos próprios componentes do sistema, é importante explicar como os componentes básicos do sistema de gerenciamento de recursos do 2K — LRM, GRM e CR — obtêm conhecimento de seus componentes dependentes.

Logo após ser iniciado, o LRM faz uma chamada à operação `resolve_initial_reference("DynamicConfigurator")` a fim de obter uma referência para o GRM de seu aglomerado. Este processo é feito por meio de mensagens *multicast* enviadas pela rede. De posse da referência do objeto, o LRM registra-se no GRM e, assim, institui a máquina onde está hospedado como participante do sistema no aglomerado.

2.1.6 Serviços e operações fundamentais

Sabemos até o momento que cada LRM deve registrar-se no GRM de seu aglomerado. No entanto, não se entrou em detalhes sobre este processo. Estes detalhes serão esclarecidos nesta seção.

Sendo essencialmente um *trader*, o GRM permite que tipos de serviços sejam incluídos, removidos ou alterados. É assim que um LRM é registrado no GRM, incluindo-se como uma oferta de serviço na sua base de dados.

Ao ser instanciado, o GRM inscreve no seu repositório de tipos alguns tipos pré-existentes e que são essenciais para o sistema. Esta característica contém uma das modificações efetuadas no *trader* original. Na iniciação do *trader* algumas propriedades fundamentais para o sistema são adicionadas à sua base de dados. Assim, o LRM registra-se no tipo designado para ele e exporta os recursos disponíveis da máquina para serem compartilhados.

O tipo utilizado pelo LRM para registrar-se é o `ResourceMgmt_Lrm`, o qual é subtipo de outro tipo, o `ResourceMgmt_base`. Este também é usado como supertipo para as ofertas de serviço ou recursos, irrestritamente, pois o identificador do `ResourceMgmt_base` é do tipo `Object` do CORBA, conseqüentemente garantindo que o identificador de um tipo seja compatível com seu tipo — veja Tabela 2.1. A idéia central nesta concepção é oferecer a possibilidade de definir qualquer serviço simplesmente tornando-o subtipo de `ResourceMgmt_base`.

Esses tipos são de extrema importância para o funcionamento do GRM e para todo o sistema de gerenciamento de recursos. A fim de prevenir que eles sejam manipulados por entidades externas ao GRM, possível através de operações publicamente expostas pelo próprio *trader*, alguns pontos foram modificados:

`remove_type(...)`: presente na interface do repositório de tipos de serviços, foi alterada para simplesmente ignorar uma chamada cujo argumento seja qualquer um dos tipos essenciais — `ResourceMgmt_base`, `ResourceMgmt_Lrm`, `ResourceMgmt_Grm` ou `ResourceMgmt_interGrm`. Desta forma, evita-se que algum cliente, seja qual for sua intenção, remova um tipo essencial do sistema.

`export(...)`, `modify(...)`: as semânticas destas operações são, respectivamente, exportar serviços e modificar serviços existentes no *trader*. Se o argumento usado na chamada da operação for `ResourceMgmt_Lrm`, todos os serviços registrados derivados de `ResourceMgmt_base` que possuem o mesmo valor da propriedade *hostname* são atualizados.

`withdraw(...)`, `withdraw_using_constraint(...)`: estas operações removem um serviço publicado no *trader*, diferenciando entre si apenas pela capacidade de retirar um conjunto de serviços de acordo com restrições fornecidas no momento da chamada. Se o argumento for `ResourceMgmt_Lrm`, as ofertas derivadas de `ResourceMgmt_base` e que possuam o mesmo valor da propriedade *hostname* são restauradas.

2.1.7 Carga e execução de componentes

Os componentes desejados pelo usuário podem ser encontrados tomando-se como parâmetro as informações contidas nos GRMs. Feita uma requisição para um determinado componente, é necessário que um mecanismo encarregue-se de encontrá-lo, carregá-lo e executá-lo. Esta seção ilustra o protocolo adotado no sistema.

Os serviços registrados e disponíveis no sistema são representados por componentes armazenados nos repositórios de componentes — *Component Repositories* (CRs). Um cliente, ao fazer uma requisição de algum serviço, receberá uma referência para o componente solicitado.

Um cliente que deseja um determinado componente, envia sua requisição para o LRM local, a qual pode ser feita através de uma interface de linha-de-comando. É possível incluir restrições para a execução como arquitetura da máquina e quantidade de recursos que usará ou o próprio nome do componente, caso saiba-se de antemão. Ao receber a solicitação, o LRM verifica se há condições de executar localmente o componente de acordo com os requisitos impostos pelo usuário e pelo próprio componente. Se estiver dentro dos limites determinados, os recursos especificados na requisição são reservados por meio do uso do DSRT, assegurando que o componente seja executado apropriadamente. Por fim, o LRM carrega o componente do repositório e o executa.

Por outro lado, se o LRM não puder atender aos requisitos especificados, ele encaminha uma mensagem com a requisição e uma referência de si mesmo para o GRM. A referência serve para que outro participante saiba de quem originou o pedido e possa retornar a referência para o componente executado remotamente em caso de sucesso. O GRM é responsável por encontrar, se possível, um candidato capaz de atender às exigências presentes no pedido. Se encontrar, a requisição é encaminhada para o candidato. No entanto, é possível que o candidato

Nome do tipo	**Identificador do tipo**
ResourceMgmt_base	IDL:omg/CORBA/Object:1.0
ResourceMgmt_Lrm	IDL:twoK/ResourceMgmt/Lrm:1.0
ResourceMgmt_Grm	IDL:twoK/ResourceMgmt/Grm:1.0

Tabela 2.1: *Identificadores dos tipos de serviço*

não consiga atender, porque, por exemplo, as informações na base de dados do GRM estavam ligeiramente obsoletas no momento da procura. Assim, o LRM adiciona um identificador (nome da máquina) num campo da mensagem especialmente criado para armazenar LRMs que não puderam satisfazer o pedido e a retorna para o GRM. Dessa forma, os candidatos já visitados são excluídos do resultado da busca na base de dados. Ilustrativamente, este processo é representado pela Figura 2.3. O passos envolvidos são descritos pelas seguintes ações:

1. um cliente faz requisição especificando as condições para execução;
2. o LRM efetua uma análise das restrições contidas na requisição e verifica se ela pode ser atendida mediante verificação da disponibilidade dos recursos na máquina;
3. se não for possível atender, encaminha a requisição para o GRM do aglomerado;
4. o GRM procura por um candidato na sua base de dados. Ele prossegue enviando a requisição para o candidato;
5. o LRM consulta a disponibilidade dos recursos pedidos através da biblioteca DSRT;
6. como o DSRT retornou uma resposta afirmativa quanto à disponibilidade dos recursos especificados, o LRM carrega a componente do repositório de componentes;
7. o LRM instancia o processo que conterá o componente;
8. o LRM onde o componente está hospedado obtém a referência do LRM que deu início ao processo de localização, que está inserida na própria requisição, e informa que a solicitação pôde ser atendida, juntamente com uma referência para o objeto para ser acessado remotamente. Isto é feito através da operação `ACK(QueryPacket)`;
9. o componente é executado.

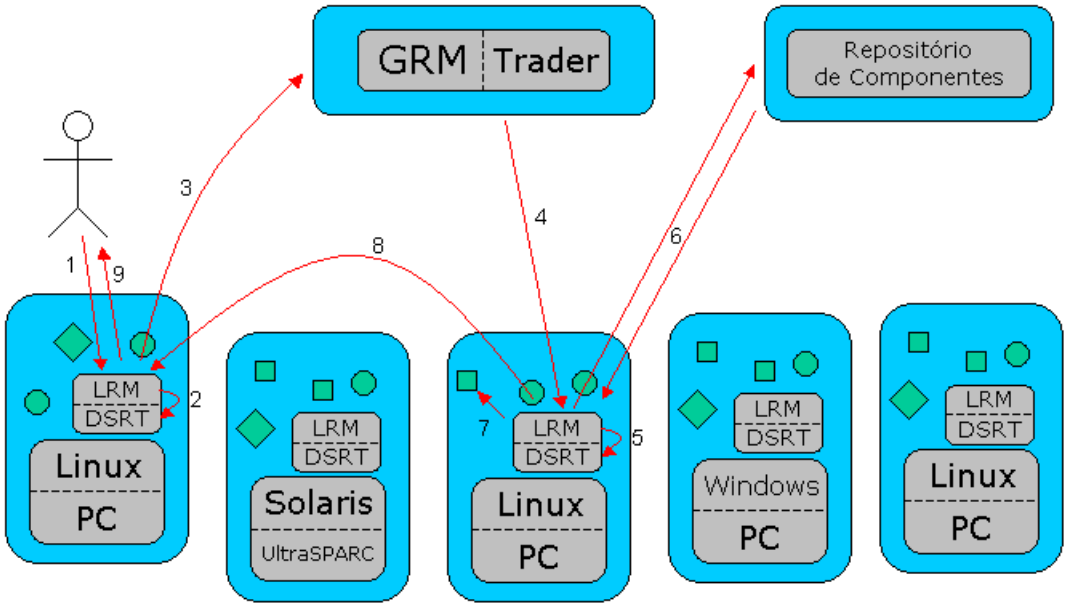

Figura 2.3: *Processo de carga e execução de componentes*

Capítulo 3

Extensão do modelo

O que foi descrito até o momento sobre o Sistema de Gerenciamento de Recursos do 2K refere-se ao modelo e implementação do sistema de gerenciamento de recursos realizado por Tomonori Yamane [Yam00] para um aglomerado isolado, ou seja, os participantes (GRMs e LRMs) estão limitados a um GRM que coordena um ou mais LRMs. Nesta dissertação, serão apresentados uma extensão das funcionalidades existentes e a agregação de outras características necessárias para o modelo, a fim de permitir que mais aglomerados sejam incorporados ao sistema, que até este ponto está confinado nos limites impostos pelas próprias características da rede ou da política de administração mantida no aglomerado. É sugerida uma ampliação, portanto. Dessa forma, pretende-se implementar a capacitação nestes conjuntos atualmente isolados para que se relacionem entre si, compartilhem seus recursos com outros aglomerados ou agreguem mais recursos, potencialmente disponíveis em outros aglomerados, para os usuários atuantes em seu domínio. Pelo mesmo motivo, toda a variedade e quantidade de recursos presentes no sistema, após estendido, passam a ser usados de forma mais racional, por meio da redução da ociosidade dos participantes.

Um dos motivos para incorporar características adicionais ao modelo torna-se claro ao levar-se em conta as restrições de abrangência do próprio aglomerado, restringindo os usuários, e por conseguinte suas necessidades, aos recursos presentes somente num único domínio (aglomerado). Permitir a comunicação entre domínios traz vantagens evidentes para os usuários e suas aplicações, proporcionadas pela possibilidade de acesso a recursos oferecidos por outros aglomerados que, conseqüentemente, enriquecem a oferta aos usuários. No entanto, também é esperado e evidente a complexidade que é produzida em decorrência disso com relação ao gerenciamento dos participantes do aglomerado. Ações como incluir, remover e manter conectados lógica e fisicamente os aglomerados entre si são importantes, pois o objetivo é permitir que a oferta de caminhos para comunicação de um participante — ou, mais precisamente, uma aplicação de usuário — não fiquem limitados a uma rede local ou a um único domínio administrativo. Todavia, inserir novas capacidades ao sistema para que os aglomerados possam ser interligados através de uma rede mais ampla e também mais suscetível a oscilações, como a Internet, implica numa variedade de obstáculos. A fim de transpô-los, são necessárias políticas e mecanismos suplementares aos existentes para garantir que o sistema trabalhe de maneira harmônica, controlada, satisfatoriamente previsível e confiável.

Ultrapassando o escopo deste projeto, existem propostas para combinar o poder computacional de grades heterogêneas, com arquiteturas e políticas distintas, permitindo que interoperem de maneira efetiva por meio de protocolos adicionais. No entanto, como descrito num artigo introdutório e genérico sobre o conceito de grades computacionais (*What is the Grid* [Fos02]), o

desenvolvimento de tais características deve estar submetido a regras comuns a todos os desenvolvedores, de modo a criar um ambiente padrão para a tecnologia de grades computacionais. O *Open Grid Services Architecture* (OGSA) [FKNT02], por exemplo, é uma especificação que representa um esforço mútuo de conceber ou conciliar estas regras entre o Projeto Globus [FK97] e inúmeras entidades do setor privado.

3.1 Arquitetura inter-aglomerado

Para que mais recursos sejam oferecidos sem que a organização e controle sobre os mesmos sejam comprometidos, são necessários novos protocolos que sejam adequados a esta nova proposta, assegurando também um quadro bastante importante: capacidade de admitir um número relativamente alto de aglomerados dispersos pela Internet, ou seja, possibilidade de ser significativamente escalável. A estrutura lógica sobre a qual os aglomerados estarão interligados é fundamentalmente crítica, pois ela influencia diretamente a escalabilidade do sistema. Uma estrutura hierárquica de árvore foi escolhida como base para comunicação inter-aglomerados devido às suas propriedades bastante favoráveis ao incremento e remoção de participantes do sistema.

Neste modelo, as funcionalidades do GRM são ampliadas ou refinadas de forma a possibilitar interação com GRMs de outros aglomerados, como ilustra a Figura 3.1. As setas tracejadas representam a comunicação entre o aglomerado e outros dois, como explicitado na Figura 3.2. Pode-se notar que o GRM é o elemento fundamental que gerencia e controla os canais de acesso a outros domínios. As ligações definidas entre cada um são decididas pelos administradores de cada aglomerado. Ainda não foram definidos modos para que um administrador possa realizar a inclusão ou remoção de um GRM pai para seu aglomerado. No momento, isto é feito simplesmente informando manualmente um IOR (referência CORBA) para o GRM. A estrutura é composta pelos GRMs interligados hierarquicamente. Com o protocolo estendido em funcionamento, cada GRM de um determinado nível na hierarquia envia mensagens de atualização contendo informações sobre o estado do aglomerado que ele gerencia para o GRM imediatamente superior na hierarquia.

A estrutura de árvore favorece a escalabilidade, pois as informações não precisam, necessariamente, serem propagadas para todos os participantes, de acordo com o protocolo desenvolvido a ser exposto adiante.

No novo modelo, cada gerenciador de recursos global (GRM) ainda mantém suas características e funcionalidades básicas originais inalteradas. Ou seja, eles ainda são responsáveis pelas informações referentes ao estado dos recursos disponíveis num aglomerado, porém podendo incorporar informações sobre recursos de outros aglomerados. No entanto, é permitido que se escolha qual será sua tarefa no sistema dentre as duas características, podendo executá-las em conjunto ou separadamente.

Conjuntamente com outros componentes, cooperando entre si, o Sistema de Gerenciamento de Recursos retira do usuário a responsabilidade de especificar a localidade do recurso desejado analogamente ao descrito no artigo de Chin e Chanson que discute o projeto e a implementação de sistemas distribuídos orientados a objeto [CC91].

Na Seção 2.1.6 foram apresentados os três tipos sob os quais trabalham os protocolos de disseminação de informações e localização e execução de componentes. Além daqueles, na arquitetura inter-aglomerado foi introduzido o tipo `ResourceMgmt_interGrm`, do qual se beneficiam as novas funcionalidades da arquitetura estendida. Sua identificação é `IDL:twoK/ResourceMgmt/interGrm:1.0`, que foi adicionada aos já existentes, presentes na Tabela 2.1.

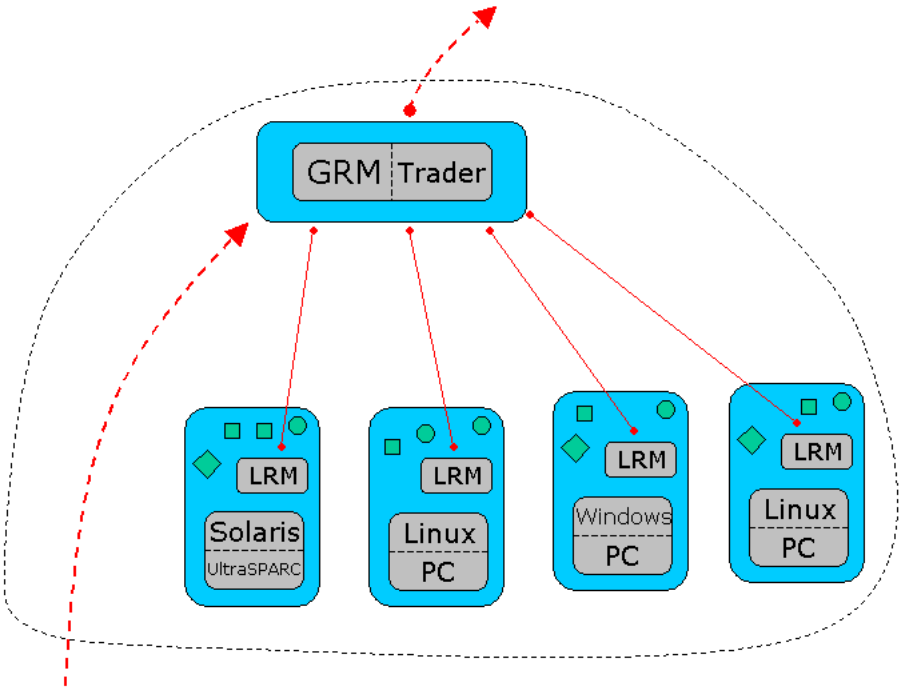

Figura 3.1: *GRM estendido*

3.2 Disseminação de informações

O subsistema de disseminação de informações estabelece que cada GRM envie atualizações periódicas para GRMs de outros aglomerados notificando o estado (disponibilidade) dos recursos de seu aglomerado ou quando uma alteração significativa no valor de um determinado recurso seja detectada. Um dos objetivos do subsistema é o uso racional e comedido dos recursos onde os GRMs estão instalados, de maneira que influenciem minimamente no consumo dos recursos que compartilham com os usuários — memória e processamento, por exemplo. Por essa razão, manter um nível relativamente baixo de tráfego de mensagens na rede, mantendo o mínimo possível de impacto na rede ocasionado pela transmissão de dados, é um ponto importante a ser solucionado pelo protocolo, que será apresentado adiante.

A Figura 3.2 apresenta uma possível configuração do sistema completo, assim como as conexões representando cada inter-relação mantida entre GRMs. A organização na estrutura de árvore resultante é feita de maneira que cada aglomerado de um determinado nível pode estar ligado a um único outro aglomerado num nível superior contíguo (aglomerado-pai), enquanto que, simultaneamente, podem haver um ou mais aglomerados conectados a ele num nível imediatamente inferior (aglomerados-filhos).

A vantagem que se obtém dessa configuração é a possibilidade de distribuir as informações pelos inúmeros nós da árvore, sendo cada um responsável por um único aglomerado diretamente e potencialmente responsável por inúmeros, no caso de possuir aglomerados-filhos. A distribuição oferece escalabilidade e evita a situação de ponto único de falha. Por outro lado, esta situação pode acontecer como decorrência da queda de algum dos GRMs que eventualmente estejam gerenciando aglomerados-filhos, porque estes ficarão inatingíveis pela outra porção do sistema, remanescente do desligamento do seu aglomerado-pai. Apesar desta situação indesejada, o maior

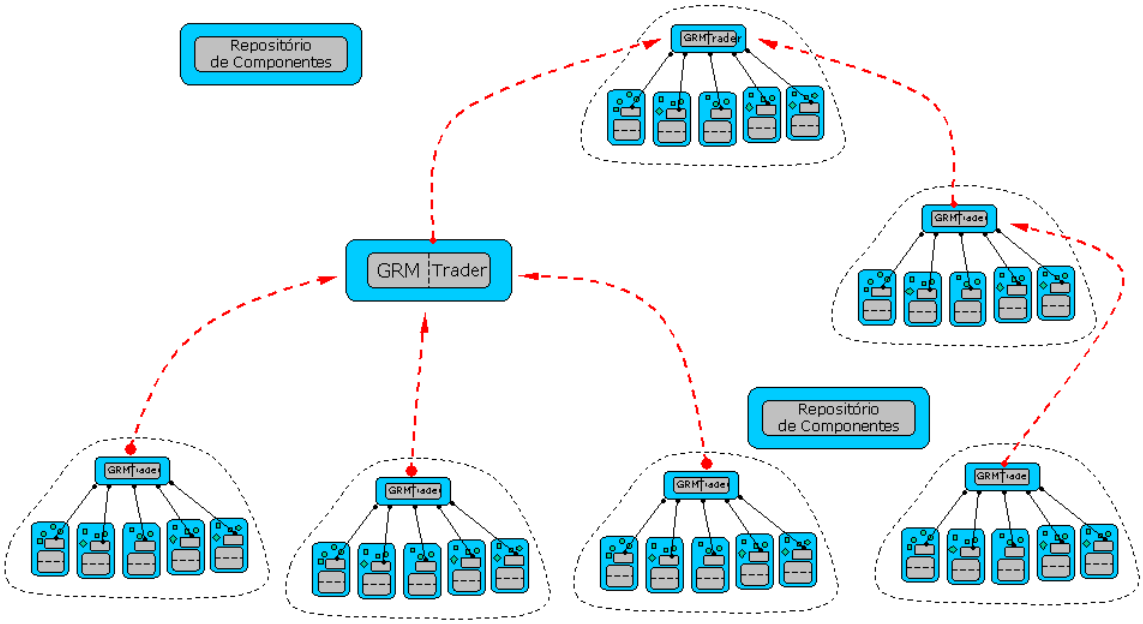

Figura 3.2: *Vínculos entre aglomerados*

prejuízo será somente a diminuição do número de ofertas de recursos no sistema.

3.2.1 Descrição do protocolo

De modo diferente do que ocorre na relação entre LRMs e o GRM do aglomerado, os dados sobre o estado dos recursos reportados por um GRM ao seu pai — GRM superior na hierarquia ao qual está ligado — não representam diretamente seus estados, mas sim uma medida *aproximada* sobre sua atual disponibilidade. Especificamente, em cada um dos GRMs internos da árvore são armazenadas informações aproximadas sobre os aglomerados da sub-árvore onde está localizado, inclusive do seu próprio aglomerado, se existirem LRMs. Ao manter informações aproximadas sobre os recursos e não os dados precisos que refletem seus estados, consegue-se diminuir substancialmente o tráfego na rede, além de permitir uma maior escalabilidade do sistema, conseqüentemente.

Cada GRM calcula a média aritmética do nível de disponibilidade de cada recurso de suas sub-árvores para então enviá-la para seu GRM pai, de forma análoga ao mecanismo usado pelos LRMs. Se o GRM também for responsável por um aglomerado local com LRMs, estes dados são computados juntamente com as informações recolhidas de suas sub-árvores. Além da média aritmética, também é calculado o desvio padrão de cada conjunto de informações de um determinado recurso. Futuramente, outras medidas também poderão ser incorporadas ao sistema com o propósito de incrementar-se a eficiência através da obtenção de candidatos cujos recursos estão mais próximos da disponibilidade exigida pela aplicação do usuário. Diferentes alternativas plausíveis para a questão são apresentadas na Seção 3.3.2.

Num cenário oposto, onde um GRM tem conhecimento total e completo do estado dos recursos nos aglomerados presentes no caminho de suas sub-árvores até os nós, seria significativo o desperdício de recursos computacionais, com o armazenamento de todas estas informações nos

aglomerados superiores. E, decerto, estas informações também trafegariam pela rede, causando outro desperdício que pode ser evitado com a proposta descrita anteriormente.

Em relação aos cenários sob os quais atualizações devem ser enviadas para seus pais GRMs, as seguintes ocasiões se aplicam:

- constatadas mudanças significativas de acordo com a métrica utilizada. Atualmente usa-se média aritmética dos valores de cada tipo de recurso, e.g. disponibilidade de memória e uso do processador;
- surgimento de outra oferta (recurso) que até aquele instante inexistia na árvore;
- o intervalo máximo sem envio de atualizações foi atingido. Esta ação tem o propósito central de notificar que aquela sub-árvore ainda está ativa. O outro é simplesmente manter as informações mais próximas da disponibilidade real dos recursos. É possível que este procedimento seja retirado, uma vez que um GRM pode detectar a ausência ou problema durante o processo de encaminhar um pedido para seu filho.

Para uma representação mais objetiva de alguns períodos freqüentemente referenciados, seguem algumas nomenclaturas utilizadas adiante:

- *probe interval* (p_1): a cada período p_1 os recursos são analisados e, se constatadas mudanças significativas em seus estados, uma notificação é enviada para o GRM pertinente. Este intervalo é aplicado na interação entre LRM e GRM;
- *update interval* (p_2): intervalo máximo para que uma notificação de mudança de estado dos recursos seja enviada ao GRM pertinente. Por exemplo, se $p_1 = 10s$ e $p_2 = 300s$, a cada $10s$ o LRM verifica o estado dos recursos nele registrados e, não havendo nenhuma mudança excedendo os limites pré-estabelecidos por $300s$, uma notificação é disparada. Este intervalo é aplicado na interação entre LRM e GRM;
- *probe interval* (p_3): análogo a p_1, com envolvimento de GRMs somente; portanto, inter-aglomerados;
- *update interval* (p_4): análogo a p_2, com envolvimento de GRMs somente; portanto, inter-aglomerados.

De uma maneira sistemática, o processo consiste nos passos logo a seguir e é disparado pela primeira vez ao iniciar um GRM. Os recursos, no princípio, estão com seus valores iguais a zero. Note que o processo é repetido indefinidamente. Ele é interrompido somente quando o funcionamento do GRM é cessado, por exemplo, através da intervenção do administrador.

1. $intervaloVerificacao \leftarrow p_3$ (intervalo para a próxima verificação)
2. $intervaloAtualizacao \leftarrow 0$ (intervalo para a próxima atualização)
3. $R \leftarrow$ { recursos registrados no sistema }
4. $M \leftarrow$ { métricas aplicadas ao sistema }
5. **Para cada** tipo de recurso r de R
6. $\quad v \leftarrow (r, m_1, m_2, \ldots, m_{|M|})$, onde $m_i \in M, 1 \leq i \leq |M|$
7. $\quad$ **Se** houve mudança significativa baseado em $m_1, \ldots, m_{|M|}$ de v, **então**
8. $\quad\quad C \leftarrow C \cup \{v\}$
9. **Se** $C \neq \emptyset$, **então**

10. Notifique as mudanças contidas em C ao GRM-pai.
11. $intervaloAtualizacao \leftarrow 0$
12. **senão**
13. $intervaloAtualizacao \leftarrow intervaloAtualizacao + intervaloVerificacao$
14. **Se** $intervaloAtualizacao > p_4$, **então**
15. Encaminhe o conjunto dos recursos registrados ao GRM-pai juntamente com as métricas.
16. $intervaloAtualizacao \leftarrow 0$
17. Aguarde ociosamente por um período especificado em intervaloVerificacao
18. Retorne ao passo 3.

O passo 3 gera o conjunto dos recursos presentes no sistema e o passo 4 contém o conjunto representante das métricas que são aplicadas aos recursos, como média aritmética e desvio padrão. No passo 5 inicia-se uma iteração para aplicar cada métrica presente no conjunto M a cada recurso do conjunto R. O resultado de cada iteração é um vetor cujos componentes são: o recurso na primeira posição seguido das métricas que foram aplicadas nele. Estes passos possuem um caráter genérico. No sistema implementado, especificamente, na parte de disseminação de informações utiliza-se a média aritmética como critério para decidir se os valores dos recursos devem ser atualizados no GRM-pai. O passo 6 expressa essa condição para que o vetor v seja incorporado ao conjunto C no passo 8. Se o conjunto C não estiver vazio, conforme condiciona o passo 9, é necessário que as novas informações sejam atualizadas no GRM-pai (passo 10). No passo 11 reinicia-se o valor de *intervaloAtualizacao*, pois as mudanças foram efetivadas no GRM-pai. No entanto, se o conjunto C estiver vazio, ou seja, se nenhum recurso teve seus valores significativamente alterados, no passo 14 a variável *intervaloAtualizacao* é incrementada com $intervaloVerificacao$, indicando o período de tempo que se passou sem que uma atualização fosse realizada. Nos passo 15 e 16, se o intervalo para atualização ultrapassou o período máximo sem atualização, os valores correntes dos recursos são atualizados no GRM-pai. No passo 17, o algoritmo indica uma pausa especificada pelo intervalo para verificação dos recursos. Finalizando, o algoritmo retorna ao passo 5 e o processo é repetido.

3.2.2 Implementação

Afora modificações nas operações existentes, foram introduzidas duas novas a fim de facilitar e dinamizar o processo de disseminação. Além disso, outras operações tiveram suas implementações alteradas para ajustarem-se ao novo ambiente com múltiplos aglomerados que se inter-relacionam, potencialmente.

- `void set_probe_interval(in long interval)`: altera o intervalo em que o LRM verifica se os recursos foram modificados, assim como seus valores. Recebe a denominação de p_1;
- `void set_update_interval(in long interval)`: altera o intervalo máximo que o LRM pode permanecer sem enviar atualizações para o GRM de seu aglomerado. Ultrapassado esse valor, o LRM obrigatoriamente requisita uma atualização com os novos valores ou recursos no GRM. Recebe a denominação de p_2;

Em relação ao gerenciador de recursos global, as entidades centrais presentes no sistema têm suas características ilustradas na Figura 3.3 e a semântica de cada operação apresentada a seguir.

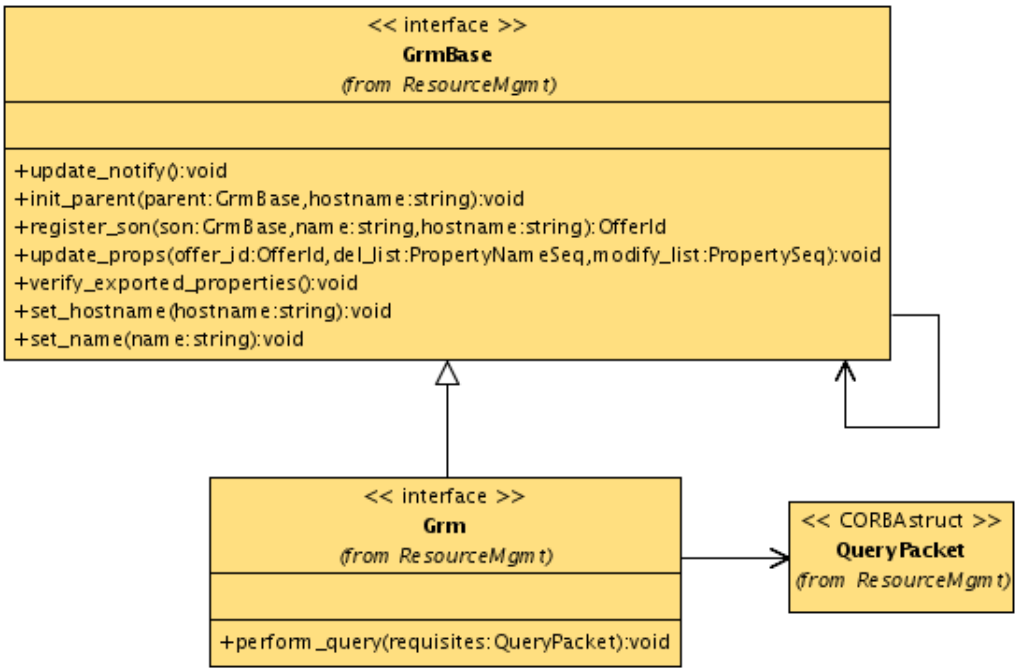

Figura 3.3: *Diagrama de colaboração do subsistema GRM*

1. `init_parent(...)`: utilizada somente pelo servidor que instancia e ativa o servente do GRM. Está na interface pública porque não é possível acessar o objeto CORBA GRM. Basicamente, ela compreende o processo de registro no GRM-pai e notifica-o para rever os valores das métricas, como a média, de seus filhos pela primeira vez.

2. `perform_query(...)`: recebe uma requisição como argumento, iniciando o processo de localização do componente nela especificado. A consulta (*query*) refere-se à solicitação do componente ou, mais precisamente, a uma tentativa de encontrar outro participante que possa atender à solicitação. Tanto LRMs como GRMs podem fazer uma chamada para esta operação. O LRM o faz ao efetuar uma requisição de um componente, basicamente. Já o GRM efetua uma chamada para encaminhar uma requisição que não pôde ser atendida dentro do aglomerado.

3. `register_son(...)`: registra o GRM como filho em outro GRM. A informação de quem deverá ser o GRM onde o registro deverá ser feito é fornecida no momento da iniciação do GRM, por linha-de-comando, ou seja, de forma estática. Esta informação é um nome de domínio. Ao iniciar um GRM, deve-se fornecer, entre outras informações, se o GRM deverá ter um pai ou não. Caso deva ter, o GRM chama esta operação para que o registro seja feito. O nome de domínio determina um participante (GRM ou LRM) no sistema e é usado para registro do objeto no servidor de nomes.

4. `set_hostname(...)`: atribui o nome do domínio associado à máquina onde está localizado o GRM. Esta operação foi concebida para substituir o nome encontrado programaticamente. Normalmente, administradores dão um nome à máquina, corretamente. No entanto, para obter o nome completo da máquina não é suficiente a operação `hostname()`

presente na maioria das bibliotecas; é necessária a concatenação do resultado obtido com o resultado de outra operação: `domainname()`.

5. `update_notify()`: por meio desta operação um GRM notifica o pai que foi realizada uma atualização nos recursos pelo filho. Dessa forma, ele calcula novamente as métricas presentes na sua base de dados a fim de saber se houve uma mudança significativa. Ela pode servir como substituta das operações de verificação periódica (p_3) do estado dos filhos e da atualização obrigatória ultrapassado o período limite (p_4). O LRM também pode utilizá-la para notificar o GRM ao qual está registrado que um atualização foi realizada.

6. `update_props(...)`: atualiza as propriedades (recursos) que estão registradas na base de dados do GRM-pai, como memória volátil disponível, espaço em disco, etc.

3.3 Localização de recursos e execução de componentes

De forma semelhante ao modo de operação do processo de execução intra-aglomerado apresentado anteriormente, no protocolo inter-aglomerado a busca é repassada para outro elemento do sistema visando adquirir mais alternativas para uma requisição, isto é, um número maior de ofertas, aumentando a probabilidade de encontrar o recurso desejado. O subsistema aqui proposto tem sua eficiência aumentada se as conexões entre pais e filhos de cada domínio (GRM e LRMs) forem estabelecidas entre aglomerados próximos, isto é, com um balanceamento satisfatório entre baixo período de latência na comunicação pela rede e a capacidade de transmissão de dados pela rede.

3.3.1 Descrição do protocolo

Primeiramente, será apresentado um panorama do protocolo de localização e execução, para efeito introdutório. Adiante, na Seção 3.3.3 será exposta uma explicação detalhada.

Ao receber uma requisição, seja de um LRM ou de outro GRM, o GRM consulta sua própria base de dados à procura de algum participante no seu aglomerado que possivelmente atenda às restrições determinadas. Caso não encontre um participante com o recurso disponível, o seguinte processo entrará em ação:

1. *Encaminha para um de seus filhos.* O GRM procura na sua base de dados uma "dica" para qual GRM filho, que representa uma de suas sub-árvores, tem maior probabilidade de poder atender à requisição; ele então encaminha a requisição para este filho. Enquanto a requisição não é atendida, o processo é repetido até que todos os nós das sub-árvores sejam considerados.

2. *Encaminha para seu pai.* Caso não possua filhos ou todas suas sub-árvores tenham sido percorridas sem sucesso, o GRM encaminha a requisição para seu GRM pai, o qual repetirá o processo descrito acima exceto, claro, para a sub-árvore de onde originou-se a requisição.

Para que as requisições não sejam encaminhadas para nós já percorridos no processo, cada aglomerado possui um nome único no sistema. No momento, um nome de domínio usual está sendo utilizado. Entretanto, essa abordagem exclui aqueles domínios que não possuem nomes válidos na Internet acessíveis por meio de serviço de nomes tradicional (DNS).

3.3.2 Métodos para escolha de candidatos

O critério de escolha de candidato a quem será encaminhada a requisição baseia-se, atualmente, em média aritmética e desvio padrão. Uma observação importante a fazer é a generalização que este método imprime na caracterização da disponibilidade dos recursos num aglomerado ou, de forma mais ampla, em toda uma sub-árvore que um aglomerado possui. Embora a média aritmética não seja um meio preciso de informar as condições dos recursos de um aglomerado, ainda assim ela constitui um bom meio. Para reforçar a análise durante a busca por uma máquina com potencial de atender as necessidades da aplicação, o desvio padrão pode ser utilizado. Com o desvio padrão, a escolha pode ser feita auxiliada pelo dado indicativo de variabilidade dos valores. Assim, comparativamente, se as médias de um recurso existente em dois aglomerados forem iguais, o desvio padrão poderia ser usado como informação decisória: um desvio padrão menor é indicativo de que aquele aglomerado, em respeito ao recurso, é mais homogêneo que o de maior desvio padrão. No entanto, o tipo de operação pode ter influência nisso. Assim, se houver uma restrição com respeito ao total disponível de memoria volátil, por exemplo "`RAM_total == 256`", o menor desvio padrão, indicativo de maior homogeneidade, seria adequado. Por outro lado, uma restrição como "`RAM_total => 256`" talvez seja fosse beneficiada por um desvio padrão maior, indicando que o valor do recurso no aglomerado está mais disperso. Claramente, se a requisição estiver exercendo restrições sobre uma variedade de recursos, o processo é mais complexo. A solução adotada consiste em aplicar o método em um só recurso. Em múltiplos recursos, o procedimento seria aplicar o método descrito para um único recurso em cada recurso presente na restrição, contabilizando os menores desvios padrões presentes em cada aglomerado e, por fim, escolhendo o participante que apresentar maior homogeneidade, ou seja, com o maior número de desvios padrões mais próximos a zero, isto é, os menores desvios padrões.

Uma abordagem similar seria o emprego de média ponderada. Ela pode ser aplicada de forma a privilegiar a escolha de um aglomerado ou sub-árvore que mantém um maior número de candidatos dispondo de um determinado recurso. O uso dessa abordagem é aceitável se considerarmos que o propósito das informações presentes nos GRMs é serem usadas como dicas simplesmente.

Outra abordagem seria calcular o intervalo [média - desvio padrão; média + desvio padrão] e verificar, para cada valor presente na requisição, se ele está presente no intervalo. Em caso afirmativo, o aglomerado associado ao intervalo configura um bom local para onde encaminhar a requisição a fim de encontrar algum participante capaz de atendê-la.

Inúmeros refinamentos podem incrementar as abordagens expostas nesta seção. Optou-se, entretanto, por fazer uma demonstração mais condensada das abordagens pelo fato de apresentarem somente pequenas disparidades entre si, não causando um impacto radical na eficiência ou acuidade de cada um.

3.3.3 O protocolo

O cenário típico do processo envolve inicialmente a requisição do componente pelo cliente e é finalizado na entrega da referência deste componente para o cliente. Ele é iniciado pelo cliente ao encaminhar o pedido para o LRM em execução na máquina e na verificação pelo LRM das condições locais para efetivação do pedido de acordo com as restrições. Não satisfeitas as condições, o pedido é encaminhado para o GRM. A partir deste ponto, recebida a requisição, o conjunto de regras abaixo define o protocolo de localização de recursos e execução do componente que é aplicado — em relação ao GRM:

1. $L \leftarrow \emptyset$
2. $L_p \leftarrow$ { LRMs potencialmente aptos a atender a requisição }
3. $L_v \leftarrow$ { LRMs já visitados } (estão presentes na requisição)
4. $L \leftarrow L_p \setminus L_v$
5. **Se** $L = \emptyset$, **então**
6. $\quad G \leftarrow \emptyset$
7. $\quad G_p \leftarrow$ { GRMs potencialmente aptos a atender a requisição }
8. $\quad G_v \leftarrow$ { GRMs já visitados } (estão presentes na requisição)
9. $\quad G \leftarrow G_p \setminus G_v$
10. $\quad$ **Se** $G = \emptyset$, **então**
11. $\quad\quad$ **Se** este GRM possui pai, **então**
12. $\quad\quad\quad G_a \leftarrow$ { nome deste GRM }
13. $\quad\quad\quad G_f \leftarrow$ { GRMs filhos deste GRM }
14. $\quad\quad\quad G_v \leftarrow G_a \cup G_f$
15. $\quad\quad\quad$ Atualize G_v na requisição recebida
16. $\quad\quad\quad$ Remova os LRMs visitados (L_v) da requisição
17. $\quad\quad\quad$ Encaminhe para o GRM pai
18. $\quad\quad$ **senão**
19. $\quad\quad\quad$ Notifique o LRM da impossibilidade de atender a requisição
20. $\quad$ **senão**
21. $\quad\quad G_v \leftarrow G_v \cup$ { nome deste GRM }
22. **senão**
23. $\quad$ Encaminhe a requisição para um LRM pertencente a L

Para melhor compreensão dos passos envolvidos no processo, segue abaixo um detalhamento de cada trecho:

1. $L \leftarrow \emptyset$
2. $L_p \leftarrow$ { LRMs potencialmente aptos a atender a requisição }
3. $L_v \leftarrow$ { LRMs já visitados } (estão presentes na requisição)
4. $L \leftarrow L_p \setminus L_v$
5. **Se** $L = \emptyset$, **então**

⋮

22. **senão**
23. $\quad$ Encaminhe a requisição para um LRM pertencente a L

Quando uma requisição é recebida por um GRM, inicialmente, é efetuada uma tentativa de atender a requisição do componente dentro do próprio aglomerado, entre os LRMs disponíveis, o que compreende o segmento acima do algoritmo.

O passo 2 consiste em extrair detalhes do pedido do cliente e usá-los como requisitos para a procura de um candidato (LRM) na base de dados do GRM. Um exemplo de requisito para ser aplicado poderia ser "`RAM_free > 50`", indicando que a memória volátil deve ser estritamente superior a 50 MB. O resultado desta consulta é o conjunto dos LRMs potencialmente aptos a atender a requisição (L_p). No entanto, para que não ocorra uma situação em que a requisição circule pelo mesmo LRM e GRM indefinidamente — é perfeitamente possível que o LRM não possa atender e retorne o pedido para o GRM — é necessário manter um registro dos participantes que já foram visitados. Este é o propósito do conjunto L_v, representado no passo 3 e que está inserido na requisição recebida pelo GRM e trafega juntamente com ela pelos nós

do sistema. O conjunto dos LRMs potencialmente adequados do qual será escolhido um para enviar a requisição é construído pela diferença do conjunto dos LRMs potencialmente aptos a atender a solicitação com o conjunto dos LRMs já visitados no aglomerado, definido no passo 4: $L \leftarrow L_p \setminus L_v$. Se este conjunto for vazio, entra em execução o processo para encontrar um candidato fora do aglomerado. Caso contrário, a requisição é conduzida para um LRM pertencente ao conjunto L.

5. **Se** $L = \emptyset$, **então**
6. $G \leftarrow \emptyset$
7. $G_p \leftarrow$ { GRMs potencialmente aptos a atender a requisição }
8. $G_v \leftarrow$ { GRMs já visitados } (estão presentes na requisição)
9. $G \leftarrow G_p \setminus G_v$
10. **Se** $G = \emptyset$, **então**

⋮

20. **senão**
21. $G_v \leftarrow G_v \cup$ { nome deste GRM }

Eventualmente, surgem ocasiões em que nenhum candidato local seja adequado para atender uma requisição. Este trecho do processo lida com essa possibilidade fazendo uma tentativa de encontrar um candidato em outro aglomerado.

De forma análoga ao método para encontrar um candidato local, é feita uma procura na base de dados por um participante externo aos domínios do aglomerado. Porém, a característica das informações utilizadas neste caso diferem sensivelmente das que são utilizadas para encontrar um LRM, cujas particularidades podem ser obtidas na Seção 3.2. De forma sintética, os dados para escolha de um GRM baseiam-se em métricas como média aritmética e desvio padrão. Assim, esses dados configuram uma "dica" e não o estado real do recursos. Verificadas as informações pertinentes e confrontando-as com as restrições contidas na solicitação, o GRM escolhe para qual aglomerado-filho (sub-árvore) deve ser encaminhada a requisição. O passo 7 sintetiza todo o processo. Atualmente, como descrito no início desta seção, os dados representando uma escolha mais apropriada da sub-árvore envolve a média aritmética e o desvio padrão de cada recurso. A média aritmética dá uma idéia aproximada de onde o recurso será encontrado e a utilização de baixos valores do desvio padrão complementa esta idéia. Construído o conjunto G_p dos GRMs aptos a atender a requisição, é necessário criar o conjunto G, retirando os GRMs por onde a requisição já passou e nos quais não obteve sucesso (G_v) dos GRMs possivelmente capazes G_p. A representação dessa operação consiste dos passos 8 e 9. Finalizando esta seqüência, verifica-se se o conjunto recém-criado G possui algum GRM e, em caso negativo, a solicitação é encaminhada para o pai (passo 10). Se ele contiver algum elemento, então escolhe-se um GRM e a solicitação é passada para ele, como mostra o passo 21.

11. **Se** este GRM possui pai, **então**
12. $G_a \leftarrow$ { nome deste GRM }
13. $G_f \leftarrow$ { GRMs filhos deste GRM }
14. $G_v \leftarrow G_a \cup (G_v \setminus G_f)$
15. Atualize G_v na requisição recebida
16. Remova os LRMs visitados (L_v) da requisição
17. Encaminhe para o GRM pai
18. **senão**

19. Notifique o LRM da impossibilidade de atender a requisição

Este trecho compreende o envio do pedido feito pelo cliente para outro aglomerado também, porém para o pai do GRM (passos 12 ao 15). Ou, se não possuir, notifica o LRM de que não foi possível atender sua solicitação (passo 19).

Na Seção 3.2 foi discutida a permanência de nós já percorridos que se tornam supérfluos porque não tiveram condições de servir a requisição. Dessa maneira, como o registro dos domínios (aglomerados) podem ser descartados, os passos 12, 13, 14 e 15 fazem exatamente isso. No passo 12, o conjunto G_a é atualizado com o GRM corrente. No passo seguinte (13), o conjunto G_f é preenchido com os filhos do GRM. No passo 14, o conjunto dos GRMs visitados recebe o GRM corrente (G_a) e são excluídos os filhos do GRM dos GRMs visitados ($G_v \setminus G_f$). Perceba que, como todo GRM efetua esta operação, a requisição conterá somente os nós visitados estritamente necessários para a continuidade da execução do protocolo de localização e execução. No passo seguinte, o conjunto que está na requisição é substituído por este novo conjunto, de forma que os antigos nós, agora desnecessários, são removidos. Mantendo somente o GRM corrente, seu pai não o considerará ao examinar as restrições da solicitação e toda a sub-árvore será descartada. Os LRMs, no passo 16, são removidos também, devido à tentativa inicial de efetuar uma procura local pelos recursos. No passo 17, a solicitação é efetivamente enviada para o pai, terminando o processo.

A Figura 3.4 apresenta um possível cenário descrevendo os passos executados pelo protocolo de localização e execução de componentes. A seguir, cada etapa é devidamente detalhada.

1. O LRM X recebe uma requisição de um cliente;

2. encaminha a requisição para o GRM A do seu aglomerado a fim de procurar candidatos para atender a requisição;

3. o GRM A não encontra um candidato apropriado e, como não possui sub-árvores, remove os nomes dos LRMs do campo pertinente na requisição e inclui seu nome na requisição. Em seguida, encaminha para o aglomerado acima na árvore, o GRM B (GRM-pai);

4. localmente, não é possível encontrar recursos que satisfaçam as restrições na requisição, tampouco na única sub-árvore que o aglomerado possui, como demonstra o campo de GRMs visitados na requisição. A alternativa é repassá-la para o GRM C, porém, antes efetuando as devidas alterações na requisição: excluir os nomes dos LRMs e adicionar seu próprio nome;

5. o GRM C, embora não tenha nenhum LRM com recursos que se ajustem ao pedido recebido, possui na sua base de dados dicas de valores dos recursos que o permitem inferir que a sub-árvore com raiz no GRM D é uma candidata apropriada, para quem a requisição deve ser encaminhada;

6. o aglomerado do GRM D também não possui um candidato, mesmo que as informações ("dicas") contidas no GRM C o levasse a inferir positivamente. Assim, ele insere seu nome e remete a requisição ao GRM C;

7. não há mais possibilidades na sub-árvore do GRM C, restando agora enviar a requisição para um nível acima dele, onde se encontra o GRM E;

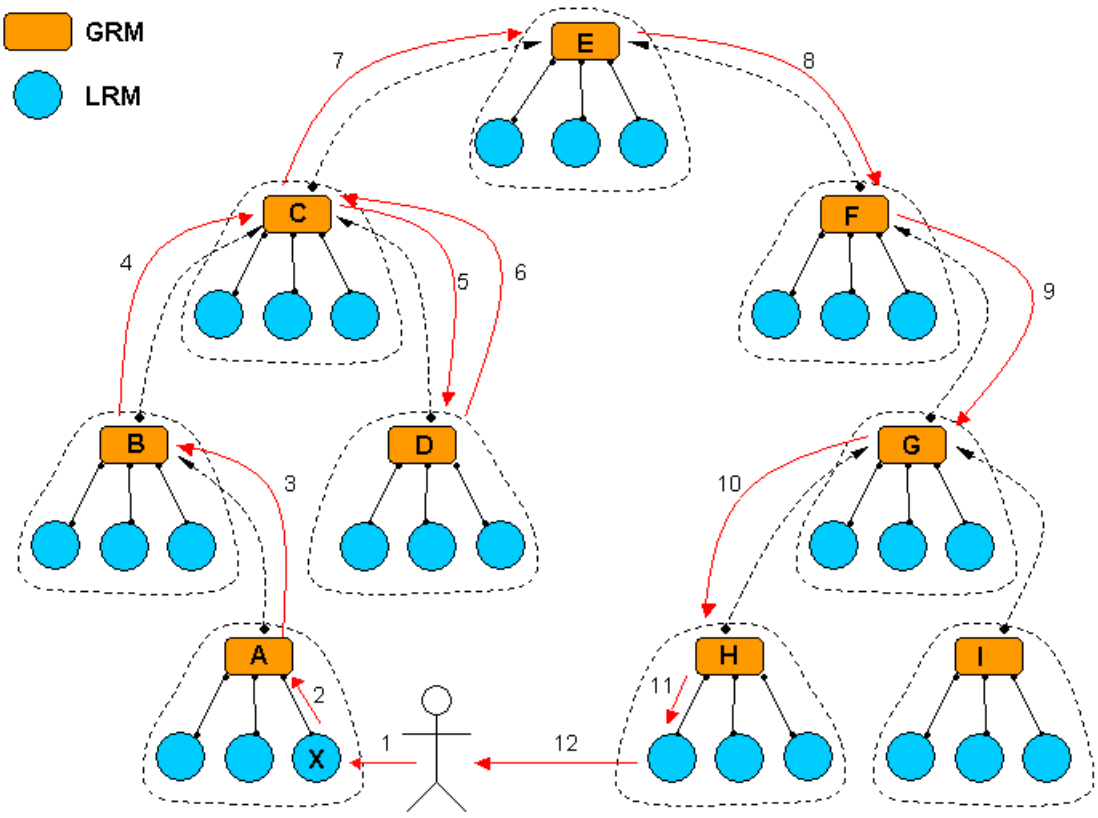

Figura 3.4: *Um cenário do protocolo de localização e execução de componentes*

8. as informações relativamente precisas do GRM E sobre os LRMs a que se interliga indicam que nenhum deles constitui um candidato. No entanto, as dicas que ele possui sobre os GRMs demonstram que é plausível encontrar os recursos solicitados nas duas sub-árvores. Como a requisição contém o GRM C na condição de GRM visitado, resta a opção de encaminhar para o GRM F;

9. o GRM F possui uma única sub-árvore, nenhum LRM apropriado e, confrontando as restrições contidas na requisição com sua base de dados, o GRM F encaminha para o GRM G;

10. pelo mesmo motivo ocorrido com o do GRM F, o GRM G encaminha sua requisição para outro aglomerado. Entretanto, ele possui duas opções razoáveis: os GRMs H e I. Havendo duas ou mais possibilidades igualmente aceitáveis, não há qualquer critério especial para escolha. Assim, a requisição é dirigida ao GRM H;

11. no GRM H existe um LRM qualificado para atender a requisição feita pelo usuário. Será ele quem verificará, em caráter definitivo, se há realmente recursos disponíveis na máquina para, em seguida, pôr em prática o que está na requisição. Isto se deve, usualmente, ao fato de que os LRMs, embora atualizem periodicamente suas informações no GRM, possuam recursos cujas condições reais não estejam de acordo com as informações na base de dados do GRM. Por exemplo, a base de dados do GRM indica que há 200 MB livres na máquina.

Porém, esta informação está de acordo com a última atualização do LRM. Neste intervalo podem ter havido alterações;

12. por fim, o LRM reserva os recursos, instancia e executa o componente, enviando uma referência (CORBA) para que o cliente o utilize.

As operações centrais empregadas no protocolo de localização de execução de componentes estão representadas na Figura 3.5. Tanto um LRM como um GRM mantêm uma operação pública chamada `perform_query()` pela qual interagem entre si para atender uma requisição, que é o argumento da operação. Ao receber a requisição, como explicado, o GRM põe em execução o processo de localização dentro do próprio aglomerado. Isto é feito através da operação `perform_query_LRM`, a qual não é pública. Se não encontrar um LRM apropriado, o GRM faz uma consulta na sua base de dados para encontrar um GRM (outro aglomerado) que esteja de acordo com as restrições presentes na requisição. Esta tarefa está representada pela operação `perform_query_GRM()` que, assim como a de procura local, não é disponibilizada para outros objetos.

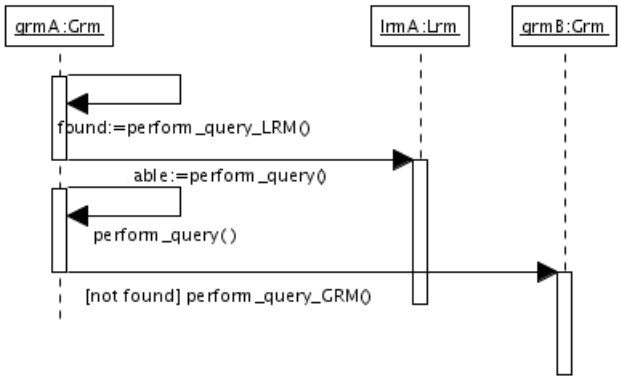

Figura 3.5: *Diagrama de seqüência do processo de localização de componente*

A Listagem 3.1 contém o código-fonte para localização dos recursos, dada uma requisição. É uma versão resumida cuja sintaxe foi alterada e em que as macros utilizadas no ACE foram retiradas, pois o intuito central é a compreensão da semântica de cada passo. Por exemplo, foram excluídos o tratamento de eventuais erros e falhas na comunicação entre uma requisição remota.

3.4 Iniciação dos serviços

Todos os GRMs, durante sua iniciação, incluem seu nome e referência num servidor de nomes único para todos os participantes. Claramente, este servidor de nomes constitui um ponto de falha que pode impedir a iniciação de outros GRMs. Embora a redundância de informações seja o tratamento ideal para esse revés potencial, ainda assim, o inconveniente pode ser minimizado adicionando-se mais servidores de nomes ao sistema configurando-os de modo a interligarem-se (federação de servidores de nomes).

Para atribuir um GRM como pai de um GRM a ser iniciado, deve-se informar seu nome no momento da iniciação, feita por linha de comando.

Em resumo, para iniciar um GRM são necessários: local onde se encontra o servidor de nomes para registrar-se; um nome arbitrário para registro; o nome do GRM ao qual ele será interligado, opcionalmente; e o número da porta à qual o servidor será associado.

3.5 Funcionalidades não implementadas

O trabalho aqui apresentado não aborda diretamente aspectos relacionados à segurança na interação entre os participantes, em qualquer grau de granularidade: objetos, subsistemas ou entre domínios. Supõe-se que todos os participantes são, desde o princípio de seu funcionamento e de maneira permanente, confiáveis e autorizados a interagirem entre si. Ainda assim, alguns requisitos são facilmente identificáveis a fim de preencher esta lacuna. Alguns pontos sobre este tema são abordados na Seção 6.1.2.

Estando além do escopo a que se destina o desenvolvimento deste projeto, não foram desenvolvidas maneiras de integrar ao sistema novos recursos, além daqueles já especificados no arquivo que descreve a interface pública do sistema. A definição, portanto, é estática somente. Uma sugestão para contornar o problema é descrita na Seção 6.1.3.

Listagem 3.1: *Trecho da operação para localização de componente (arquivo Grm_i.cpp)*

```
template <class TRADER_LOCK_TYPE, class MAP_LOCK_TYPE>
void
Grm_i<TRADER_LOCK_TYPE, MAP_LOCK_TYPE>::
perform_query(const ResourceMgmt::QueryPacket &requisition) {
  ResourceMgmt::QueryPacket req = requisition;
  CosTrading::OfferSeq_var offers;

  ACE_TRY {
    // Primeiramente, procura-se um LRM dentro do aglomerado
    offers_LRM = perform_query_LRM(req);

    if (offers->length() == 0) {
      // Procura por uma sub-árvore capaz de atender
      offers = perform_query_GRM(req);
      if (offers->length() == 0) {
        // Envia a requisição para o pai, se possuir
        // O pai foi atribuído a "grm_parent_" na iniciação do GRM
        if (CORBA::is_nil(this->grm_parent)) {
          // Notifica o LRM de que não foi possível atendê-lo
          ResourceMgmt::Lrm_var lrm =
            ResourceMgmt::Lrm::_duplicate(req.sender.in());
          lrm->NACK(req.sem_tag);
        } else {
          // Atualiza os nós visitados
          update_visited_nodes(req, this->hostname);
          grm_parent_->perform_query(req, ACE_TRY_ENV);
        }
      } else {
        // Encaminha a requisição para um GRM-filho
        ResourceMgmt::Grm_var grm =
          ResourceMgmt::Grm::_narrow(offers[0].reference.in());
        grm->perform_query(req, ACE_TRY_ENV);
      }
    } else {
      // Repassa a requisição para um LRM encontrado
      ResourceMgmt::Lrm_var lrm =
        ResourceMgmt::Lrm::_duplicate(req.sender.in());
      lrm->perform_query(req, ACE_TRY_ENV);
    }
  }
}
```

Capítulo 4

Análise de desempenho

A análise de desempenho objetiva extrair dados representativos sobre o comportamento do sistema durante seu funcionamento em situações diversas. Ao utilizar-se de procedimentos sistemáticos para obter essas informações, é possível inferir de maneira segura se o sistema está de acordo com as expectativas de desempenho, além de fornecer instrumentação para diagnosticar se o sistema merece um ajuste em suas características de forma a adquirir um desempenho mais satisfatório, ou ainda, comparar alternativas de possíveis configurações. Torna-se plausível, a partir de técnicas aplicadas à análise de desempenho, a caracterização do sistema como um todo ou em suas partes constituintes, dependendo da granularidade e objetivos desejados, sendo possível predizer o comportamento em situações distintas e ambientes variados.

Inicialmente, serão apresentados os passos principais para conduzir um processo sistemático de análise de desempenho de um sistema, seguido de sua aplicação ao sistema descrito apresentado nesta dissertação. Posteriormente, as métricas serão apresentadas e serão descritos a infra-estrutura utilizada e os métodos utilizados para extraí-las do ambiente e armazená-las.

4.1 Uma abordagem sistemática para avaliação de desempenho

Raj Jain, no livro *The Art of Computer Systems Performance Analysis*[Jai91], define alguns passos que podem ser aplicados a um leque variado de sistemas, ainda assim respeitando suas diferenças intrínsecas, em maior ou menor grau de intensidade. Descrevendo-os de forma sintetizada:

1. *Definição dos objetivos e descrição do sistema:* é necessário definir o sistema a ser estudado e o que o constitui delineando seus limites. Define-se o que é desejado da análise em relação ao sistema submetido à analise;

2. *Descrição dos serviços e dos propósitos:* definir as funcionalidades que o sistema deve oferecer, assim como seus propósitos. Com isso em mãos, a escolha das métricas e cargas a serem escolhidas posteriormente torna-se mais adequada;

3. *Seleção das métricas:* por meio da métrica, selecionamos o critério de comparação do desempenho. Como exemplos, podemos citar a disponibilidade dos serviços oferecidos pelo sistema, a velocidade da rede (vazão) e a latência;

4. *Seleção dos parâmetros:* a lista de todos os parâmetros que afetam o desempenho;

5. *Seleção dos fatores a serem estudados:* os fatores a serem estudados são os parâmetros escolhidos anteriormente e que exercem alguma influência no valor das métricas, ou no sistema em si. Os parâmetros que variam são denominados *fatores* e suas condições denominados *valores*. Os parâmetros que não são classificados como fatores, são classificados como constantes;

6. *Seleção da técnica de avaliação:* seleciona-se o método a ser utilizado de acordo com os recursos disponíveis e nível desejado de acurácia, por exemplo;

7. *Seleção da carga a ser aplicada:* consiste em escolher um conjunto de requisições a que o sistema será submetido, como o número de componentes solicitadas durante um determinado período;

8. *Elaboração dos experimentos:* de posse dos fatores e seu níveis estabelecidos *a priori*, é necessário elaborar os experimentos que fornecerão o maior número de informações com esforços aceitáveis;

9. *Análise e interpretação dos dados:* análise estatística é um instrumento de análise que permitirá, a partir dos resultados dos dados processados, interpretá-los.

10. *Apresentação dos resultados:* apresentar por meios gráficos, os resultados obtidos no passo anterior, tornando mais claro e objetivo o que se pretendia com o experimento e o que foi descoberto em relação ao sistema.

4.2 Aplicação dos passos

O processo aplicado aos experimentos foi elaborado de acordo com os itens anteriormente descritos. Abaixo segue a utilização dos passos para definir cada parte constituinte dos experimentos.

4.2.1 Definição dos objetivos e descrição do sistema

O sistema a ser avaliado consiste primordialmente de dois subsistemas: o sistema de atualização de informações e o sistema de localização e execução de componentes. O objetivo do experimento é obter e agrupar informações para posterior indução sobre a escalabilidade do sistema e para fornecer indicativos sobre sua robustez.

4.2.2 Descrição dos serviços e dos propósitos

Os subsistemas possuem as seguintes características e propósitos:

- *Sistema de disseminação de informações:* responsável pela disseminação de informações sobre o estado dos recursos dos participantes do sistema, especificamente os GRMs. Cada participante encarrega-se de atualizar os dados em seu pai, quando necessário, podendo propagar-se até o topo da árvore que constitui o sistema completo.

- *Sistema de execução de componentes:* o protocolo que governa este sistema oferece meios de localizar e executar um componente localizado na "região" de algum GRM, de acordo com requisitos impostos pelo usuário ao definir os parâmetros de busca desejados. Se o componente não for encontrado devido às restrições que não puderam ser satisfeitas, o LRM que efetuou o pedido recebe uma mensagem condizente. Caso contrário, o componente é

devidamente executado na máquina local, no aglomerado local ou até mesmo em um aglomerado remoto.

4.2.3 Seleção das métricas

A fim de identificar possíveis comportamentos indesejáveis e, certamente, desejáveis das funcionalidades do sistema, algumas métricas foram escolhidas com o propósito de construir os fundamentos para inferências coerentes. São elas:

- *Número de mensagens enviadas e recebidas por cada participante*

  O número de mensagens enviadas demonstra o quanto as informações a serem disseminadas podem ser dispersas por todo o sistema sem que o poder de decisão ou a representação dos outros nós, principalmente as sub-árvores, fiquem comprometidos.

- *Utilização relativa do processador pelo participante*

  Cada GRM tem sua utilização na máquina inspecionada periodicamente. Detalhes sobre o mecanismo de coleta serão esclarecidos adiante.

- *Quantidade de memória total utilizada pelo participante*

  Assim como o uso do processamento, também é coletado o consumo de memória de cada GRM ao longo de cada experimento.

- *Quantidade de chamadas de atualização das informações em cada participante*

  A cada notificação de que uma alteração na quantidade de recursos disponíveis foi efetuada, uma variável armazena o número total de chamadas que foram feitas. No final do experimento, portanto, temos o número de vezes que foi acionada a operação de verificação dos recursos disponíveis naquele participante. Esta métrica destina-se a avaliar o protocolo de disseminação de informações.

- *Latência na requisição de um componente*

  Quando uma requisição é feita por algum cliente, um pacote com os dados informativos daquela requisição é criado por um LRM e os participantes passam a emiti-lo entre si sob a gerência do protocolo de execução de componentes. A métrica aqui relatada simplesmente corresponde ao intervalo gasto a partir da emissão da requisição pelo cliente até a execução do componente desejado.

- *Número de bytes inseridos na rede*

  Uma solicitação para atualização dos dados gera tráfego na rede, pois é uma chamada remota. Pretende-se coletar informações sobre o número de bytes emitidos para efetuar esta operação.

A partir destas métricas apresentadas, podemos obter um retrato satisfatório do comportamento do sistema, mesmo não constando alguns dados que exercem influência pouco relevante ao seu funcionamento apropriado.

4.2.4 Seleção dos parâmetros

Nesta seção serão listados os parâmetros que afetam o desempenho do sistema. Certamente, durante as análises, houveram inclusões e remoções, de forma a manter mais próximo das necessidades do experimento, dentro da viabilidade e limitações de execução dos mesmos.

Conforme apresenta Raj Jain [Jai91], essa lista pode ser dividida em parâmetros relacionados ao sistema e parâmetros relacionados à carga aplicada ao sistema durante o experimento. No primeiro enquadram-se parâmetros que geralmente não variam, como plataforma de software e hardware, enquanto que na segunda categoria, enquadram-se, por exemplo, requisições de usuários, as quais possuem probabilidade considerável de variar em circunstâncias diferentes. Representando a primeira categoria, a dos parâmetros de sistema, foram selecionados:

- o número de participantes;
- "velocidade" do processador;
- a arquitetura e o sistema operacional;
- capacidade da rede — grande área, no caso, a Internet.

Como parâmetros da carga aplicada, foram selecionados:

- número de atualizações simultâneas, ou seja, quando um processo de atualização é iniciado no momento em que o sistema (GRM) está lidando com outra anterior;
- quantidade de informações para serem atualizadas;
- carga na rede;
- carga na máquina - processador e memória;
- largura de banda;
- taxa de perda de pacotes na rede entre participantes diretamente associados.

4.2.5 Seleção dos fatores a serem estudados

Na seção anterior foram escolhidos os parâmetros que exercem impacto significativo no desempenho do sistema. Nesta seção, serão selecionados aqueles que deverão ter suas variações nos experimentos estudadas com maiores detalhes. Outros também poderiam ser escolhidos para incorporar esta lista, entretanto, não acrescentariam diferenças notáveis nos resultados em relação aos aqui já apresentados.

- Rede local e rede de grande área (Internet);
- Número de LRMs participantes no sistema;
- Número de GRMs participantes no sistema;
- Carga na máquina.

O número de participantes é fator fundamental para o objetivo do experimento. Por meio da variação deste fator em combinação com os demais, é possível analisar o comportamento do sistema à medida que mais participantes são incorporados, aferindo a escalabilidade do sistema.

Para efeito de simulação das atividades de um usuário, a carga em cada máquina foi gerada sinteticamente por meio de um procedimento simples, o qual será descrito adiante.

Um aspecto importante a ser destacado refere-se à obtenção da utilização do processador por parte da entidade estudada — LRM ou GRM. A coleta desta métrica nos experimentos é feita periodicamente lendo os dados necessários para calcular a utilização do processador. Outro modo, significativamente diferente, consiste em verificar o tempo de permanência em estado de execução do programa — real, usuário e sistema. De posse deste valor e sabendo-se o período de duração do experimento, podemos calcular uma taxa que apresentaria uma situação pouquíssimo impactante ao ambiente, isto é, um valor baixo de utilização. Com o intuito de obter-se, inclusive, eventuais discrepâncias, especificamente picos, de valores em relação à média, optou-se pelo registro amostral deste parâmetro em intervalos constantes ao longo do experimento.

4.2.6 Seleção da técnica de avaliação

A técnica escolhida para avaliação foi a de medição, que consiste em monitorar determinados estados do sistema submetendo-o a uma carga pré-estabelecida e de forma controlada.

4.2.7 Seleção da carga a ser aplicada

A carga consiste de um conjunto de requisições feitas ao sistema com um determinado propósito, ou ainda, como forma de induzir um distúrbio no ambiente a fim de diagnosticar o comportamento do sistema em resposta a estes fatores externos alterados. Trabalhos como [Jai91], [Fei02] e [CS93] foram utilizadas para elaborar o modelo e implementar a carga a ser aplicada.

Para os dois protocolos implementados, a carga aplicada durante a execução difere consideravelmente, pois, ainda que sejam dependentes em alguns pontos, é possível manter uma perspectiva diferente de cada serviço oferecido.

- *Sistema de disseminação de informações:* pelo fato deste sistema efetuar sua tarefa de atualização somente quando há uma variação significativa de algum recurso (os limites são pré-estabelecidos), uma carga adequada ao objetivo do experimento deve simplesmente causar um distúrbio nos recursos. Assim, o recurso selecionado para sofrer a alteração induzida de estado foi a utilização relativa de processamento da máquina. No entanto, somente as máquinas onde existem LRMs são expostas a estes distúrbios, pois são eles que disparam o processo de atualização dos recursos.

- *Sistema de execução de componentes:* para este sistema foram criadas algumas pequenas aplicações, as quais simplesmente fazem requisições freqüentes de execução de componentes ao LRM da máquina onde está sendo executado. Detalhes deste sistema podem ser encontrados no Apêndice C.

4.2.8 Elaboração dos experimentos

Com o propósito de conhecer os efeitos exercidos pelos fatores selecionados, nos seguintes subitens são descritos como os experimentos foram elaborados. Os experimentos estão personalizados e ajustados para cada subsistema individualmente.

As máquinas que serviram de ambiente para os experimentos continham as seguintes características:

- arquitetura PC;
- processadores Intel ou AMD;
- sistema operacional GNU/Linux;
- acesso à rede por uma interface Fast Ethernet, cujo limite nominal é de 100Mbps;
- valores BogoMips [bog] variando aproximadamente entre 1.400 e 5.200 — com a ressalva de que BogoMips é uma métrica pouco precisa, embora sirva para construir uma idéia aproximada do poder computacional do equipamento.

Os experimentos, embora tenham sido bastante elucidativos, poderiam oferecer amostras mais próximas do real efetuados em ambientes com maior oferta de recursos. Este ponto pode ser considerado como uma tarefa futura. A plataforma PlanetLab [PACR02; BBC+04], um ambiente de testes em escala global, é uma potencial candidata para implementação desta idéia. Maior precisão quanto às características de cada máquina utilizada nos experimentos pode ser encontrada no Apêndice D.

Foram realizados outros experimentos além dos apresentados a seguir. Aqueles distinguiam-se destes pelo número de participantes e altura da árvore, principalmente. No entanto, eles não são apresentados pelo fato de que as informações colhidas foram bastante similares aos experimentos discutidos nesta dissertação. Cabe destacar que, durante o curso dos experimentos, algumas máquinas poderiam ficar indisponíveis por razões diversas. Se esta eventualidade ocorresse com algum GRM, o experimento era repetido.

Sistema de disseminação de informações

A elaboração de cada experimento é discutida e detalhada nos próximos itens. Ao todo, são detalhados dois experimentos relevantes.

Experimento 1

Este experimento visa colher dados relevantes no funcionamento do sistema de disseminação de informações. Os dados colhidos (e.g. consumo de memória e uso do processador pelos GRMs) são analisados posteriormente para inferir alguns aspectos do sistema em execução, como sua escalabilidade, o impacto que ele gera no ambiente onde é executado e o impacto gerado em cada máquina que possui um GRM instalado.

Para aproximar-se de um ambiente mais real, a altura da árvore neste experimento constitui 3 níveis (Figura 4.1). Os LRMs tiveram o valor de seu período de verificação (p_1) — intervalo entre cada diagnóstico dos recursos disponíveis no sistema — atribuído para $5s$ e o período de atualização (p_2) — período máximo para uma nova notificação seja enviada ao GRM — para $10s$, num ambiente em que a variação dos recursos disponíveis era irrisória. A escolha destes baixos valores para os períodos é fundamentada no cenário atípico que eles representam, estando mais próximos de uma situação de pior caso. O segundo experimento, descrito a seguir enquadra-se num cenário comum, em que os valores são mais esparsos.

Experimento 2

Os tipos de informações colhidas neste experimento são as mesmos citados no experimento anterior, assim como seu objetivo de mensurar a escalabilidade e o impacto do sistema de disseminação de informações. No entanto, neste experimento os períodos p_1 e p_2 foram alterados e uma carga no processador foi gerada artificialmente representando sua utilização por usuários, conforme descrito a seguir.

Os LRMs tiveram o valor de seu período de verificação (p_1) atribuído para $30s$ e o período de atualização (p_2) para $180s$. Neste experimento houve a imposição de uma carga criada sinteticamente obedecendo uma distribuição exponencial com média de $60s$. Mais detalhes podem ser encontrados no Apêndice B.

Em ambos os experimentos, a altura da árvore de GRMs formada pelo sistema foi 3. Foi realizado um experimento com altura dois, no entanto, os resultados não foram suficientemente expressivos para serem detalhados aqui. A mesma configuração do experimento 1 foi adotada neste experimento, a qual está ilustrada na Figura 4.1. Nela são mostradas as três localidades onde os experimentos foram executados: nos computadores da Universidade Rey Juan Carlos, em Madrid, Espanha, nos computadores do IME-USP e na máquina denominada "grm2", que é meu computador pessoal, conectado à Internet através de ADSL.

Baseado nos intervalos acima especificados, é possível impor ao sistema uma situação atípica que obriga a atualização de informações e, por conseguinte, verificar que sua eficiência não se deteriorará ao longo do tempo de funcionamento em condições reais.

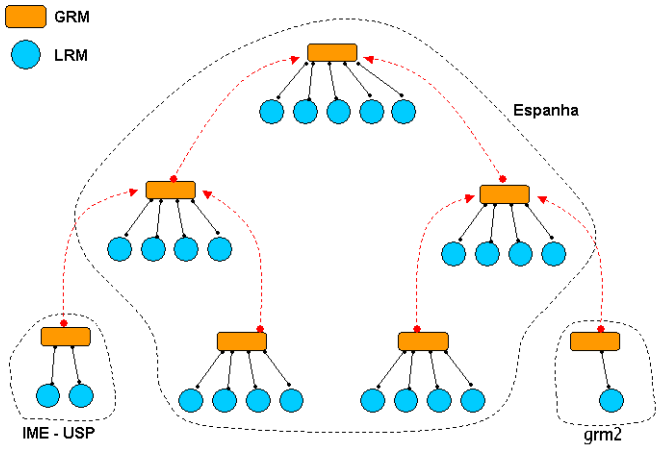

Figura 4.1: *Configuração para os experimentos 1 e 2*

Sistema de execução de componentes

Neste experimento, procura-se adquirir detalhes sobre o funcionamento do sistema de execução de componentes, especialmente o tempo decorrido entre a realização de uma requisição e o início da execução do componente solicitado, sob condições em que não há controle do dinamismo do ambiente.

Experimento 3

Neste experimento, o sistema de localização e execução de componentes é o foco das observações, ainda que, concomitantemente, o sistema de disseminação de informações entre os aglomerados também esteja em funcionamento. O objetivo deste experimento foi analisar o comportamento do sistema mediante inúmeras requisições feitas por usuários. A disposição dos GRMs e LRMs usados neste experimentos podem ser vistos na Figura 4.2. Todos os computadores usados no experimentos estão localizados na Universidade Rey Juan Carlos, na Espanha.

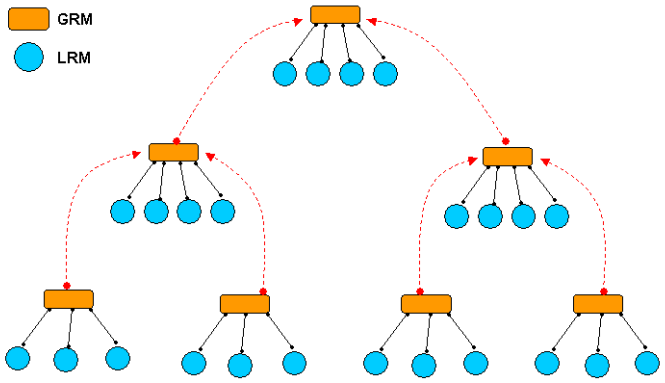

Figura 4.2: *Configuração para o experimento 3*

Foram mantidos os intervalos para verificação do estado dos recursos (p_1) e do período limite para atualizar as informações nos GRMs pelos LRMs (p_2). Neste experimento, os períodos correlatos referentes aos GRMs foram desativados, ou seja, a verificação de mudança significativa nos recursos dos LRMs (p_3) presentes na base de dados e do prazo máximo para notificar o GRM-pai permaneceram inativos. No entanto, estas verificações feitas pelo GRM eram disparadas por um LRM ao atualizar a base de dados do GRM de seu aglomerado. Espera-se que a utilização do processador seja ligeiramente mais alta por causa desta abordagem, pois cada LRM avisará o GRM de seu agrupamento quando uma atualização for efetuada. Adiante, na Seção 4.6, veremos que a utilização do processador foi realmente baixa.

A tarefa de simular um usuário foi de responsabilidade de um programa desenvolvido exclusivamente para este propósito, chamado `execution_experiment`, o qual efetua requisições sucessivas ao LRM local. Os detalhes de seu funcionamento podem ser encontrados no Apêndice C. Para este experimento, particularmente, ele sofreu uma modificação, de modo a alternar os intervalos da restrição $x < \texttt{RAM_free} \leq y$, onde x e y são valores modificados a cada requisição efetuada incrementando seus valores em 10Mb em relação à anterior. Na primeira iteração, o intervalo foi $0 < \texttt{RAM_free} \leq 10$, na segunda passou a ser $10 < \texttt{RAM_free} \leq 20$ e assim por diante. Entretanto, um último intervalo foi estipulado baseado na memória volátil total instalada de aproximadamente 512 MB: $390 < \texttt{RAM_free} \leq 400$. A Listagem 4.1 ilustra o processo — as linhas irrelevantes para o entendimento foram retiradas.

Listagem 4.1: *Trecho do programa para efetuar requisições de componentes (arquivo ExperimentExecution.cpp)*

```
for (;;) {
  static int lower_interval = 0;
  const int interval = 10;
  static const CORBA::ULong service_constraint_length =
    ACE_OS::strlen("RAM_free > xxx and RAM_free <= yyy");
  static char service_constraint_temp[service_constraint_length + 1];
  ACE_OS::sprintf(service_constraint_temp, "%s%d%s%d",
                  "RAM_free > ", lower_interval,
                  " and RAM_free <= ", lower_interval + 10);
  service_constraint = CORBA::string_dup(service_constraint_temp);
  // as máquinas possuem aproximadamente 512MB
  lower_interval = (lower_interval + interval) % 400;

  // efetua a requisição
  experiment.run(comp_name, service_constraint,
                 preference, specified_props, category_name);

  CORBA::string_free(service_constraint);

  // aguarda um período obtido por meio de uma distribuição exponencial
  int idle_interval = ACE_static_cast(int, rint(next_call(mean)));
  ACE_OS::sleep(idle_interval);
}
```

A média utilizada para o cálculo dos intervalos do programa que efetua as requisições foi de $5s$. Assim, o intervalo de memória volátil livre retorna aos seus valores limites iniciais a cada $200s$, aproximadamente.

4.3 Coleta das métricas

- *Número de mensagens enviadas e recebidas por cada participante*

  As mensagens enviadas e recebidas aqui tratadas são as requisições para notificação de atualização dos dados de cada recurso disponível. Para armazenar o número de mensagens enviadas e recebidas, existe, simplesmente, um contador para esta finalidade num método de um objeto GRM. Assim, a cada chamada deste método, esta variável estática é incrementada e, ao término do experimento, ela é consultada.

- *Utilização relativa do processador pelo participante*

  Um pequeno sistema foi criado para este propósito (vide Seção 4.4), a fim de facilitar a centralização dos dados coletados e para tornar menos trabalhosos o tratamento e a manipulação dos dados.

- *Quantidade de memória total utilizada pelo participante*

  O mesmo sistema encarregado de colher o dado descrito no item anterior foi utilizado para averiguar periodicamente a quantidade de memória total utilizada por um participante.

- *Quantidade de chamadas de atualização das informações em cada participante*

  Esta informação é obtida pelo incremento sucessivo de uma variável estática situada num método do GRM que inicia o procedimento de atualização das informações. O valor do número de chamadas feitas é escrito num arquivo em disco para posterior análise.

- *Latência na requisição de um componente*

  O programa encarregado de cumprir as requisições dos componentes possui um mecanismo para calcular o tempo decorrido entre o início da requisição e seu término, seja em caso de atendimento da requisição ou em falha.

- *Número de bytes inseridos na rede*

  O programa usado para esta tarefa é o Ethereal [Teaa]. Suas características permitem que a pilha do protocolo de comunicação seja detalhadamente analisada em função de cada camada, de maneira a informar, entre outras funcionalidades, o comprimento da mensagem, do pacote ou do quadro enviado pela interface de rede presente na máquina.

4.4 Sistema de coleta de informações de desempenho

A coleta de informações possui relevância especial para a análise de desempenho. Sem tal sistema, a tarefa de obter os dados sobre os recursos observados seria um processo bastante moroso e suscetível a erros, pois várias máquinas constituiriam os experimentos, somando assim, um esforço extra para centralizar e processar os dados coletados. Para operacionalizar a tarefa foi concebido um pequeno sistema para exercer integralmente a função de verificação periódica e armazenamento dos dados coletados. Os requisitos centrais no qual se baseou o sistema foram: consumo de memória que não causasse grande impacto em relação ao total de memória de uma determinada máquina e também pouco impacto no consumo de processamento.

O sistema foi compilado vinculando-o a bibliotecas estáticas com o propósito de diminuir as incompatibilidades, ou até mesmo a inexistência, de algumas bibliotecas dinâmicas das quais depende quando distribuído pelas máquinas onde serão utilizados.

Suas funcionalidades centrais são:

- coletar os dados de algum recurso disponível na máquina. É possível expandir de forma a diversificar os recursos monitorados, por meio de implementação de uma interface contendo métodos básicos essenciais.

- armazenamento dos dados obtidos de forma persistente. Em colaboração com os objetos responsáveis pela funcionalidade do item anterior, a persistência dos dados é feita por objetos destinados a interagir com meios de armazenamento de informações. Foram feitas implementações para armazenamento em banco de dados e arquivo (texto). Não foram criadas classes com alto grau de abstração em razão da complexidade desnecessária que seria inserida no sistema, além de potencialmente ir de encontro a alguns dos requisitos do próprio sistema de coleta, como o tamanho a ser ocupado em memória.

O sistema foi implementado em C++, como forma ponderada de atingir o objetivo de consumir memória de maneira razoável e ser desenvolvido com base no modelo orientado a objetos.

O diagrama de classes representando o sistema é apresentado na Figura 4.3. As responsabilidades atribuídas a cada classe e as operações públicas de cada uma estão a seguir:

- *Monitor:* abstração de um elemento que monitora um determinado recurso ou variável dinâmica do sistema em estudo. Outras classes devem ser sub-classes desta se é desejado obter especialização e adição das suas operações básicas, assim como faz *MemoryMonitor*.

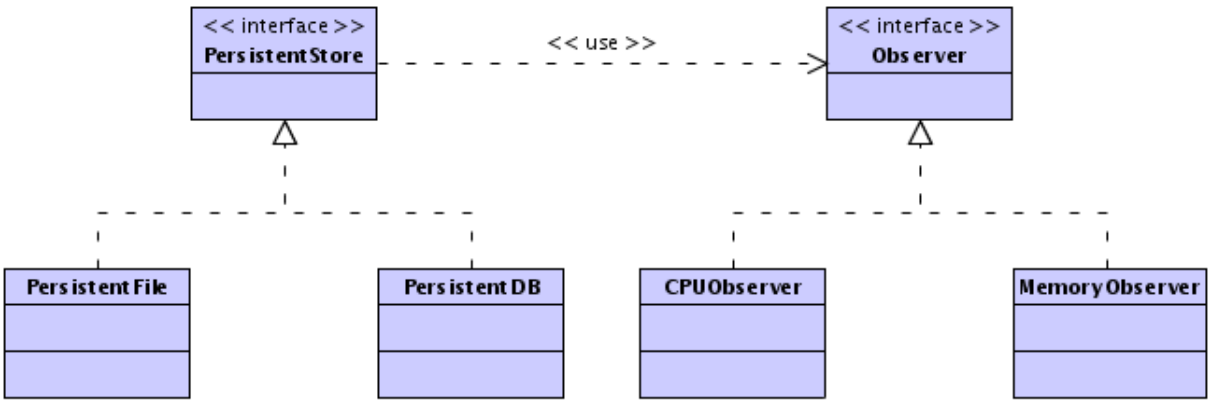

Figura 4.3: *Diagrama de classes do monitor de recursos*

- *MemoryMonitor* e *CPUMonitor:* ambas são sub-classes da anterior e possuem a responsabilidade de coletar o valor da memória consumida e do processamento, respectivamente. Estes valores são referentes a um único processo, portanto, é necessário informar a identificação do processo pelo sistema operacional (*pid*) — no momento da construção. Operações públicas:

```
virtual double collect() = 0;
static double collect(pid_t pid);
virtual double getValueCollected() = 0;
```

- *PersistentStore:* interface que possui um único método, cuja semântica é armazenar o valor obtido por um *Monitor* na mídia correspondente, de acordo com a especialização feita pela classe que a implementa. Operação pública:

```
virtual void store(Monitor &monitor, const char *subsystem) = 0;
```

Onde:

- *PersistentDB* e *PersistentFile:* implementam a interface *PersistentStore* para armazenar os dados obtidos através de um *Monitor* num banco de dados e num arquivo, respectivamente. O banco de dados utilizado para armazenagem foi o PostgreSQL [Teab]. A Figura 4.4 detalha as entidades utilizadas. Não foi criada uma camada com maior abstração, a fim de interagir com diferentes bancos de dados, pelos motivos supra-citados. Operações públicas:

```
PersistenceDB(const char *host,
              const char *dbName,
              const char *user,
              const char *password,
              const char *experiment);

PersistenceDB(const char *localHostname,
              const char *host,
              const char *dbName,
              const char *user,
              const char *password,
              const char *experiment);

virtual void store(Monitor &monitor, const char *subsystem);
```

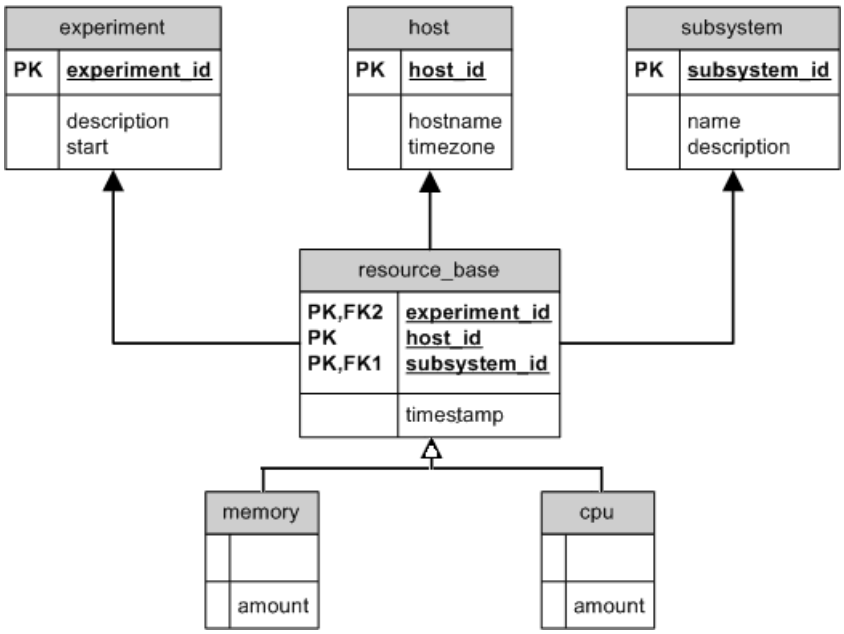

Figura 4.4: *Diagrama Entidade-Relacionamento do banco de dados para experimentos*

4.5 Sistema para geração de carga

Basicamente, dois sistemas para geração de carga foram criados, ambos destinados a avaliar os sistemas de disseminação de informações e de execução de componentes. Detalhes do funcionamento e concepção destes sistemas são objetos desta seção.

4.5.1 Carga para consumo de processamento

Um pequeno programa, escrito em C por simplicidade, foi criado para simular o consumo de processamento de uma máquina ocasionado por aplicações de que um usuário se utiliza normalmente. O código-fonte encontra-se no Apêndice B.

Basicamente, ele consiste de uma função que realiza um laço (*busy wait*) durante um determinado período. A alternância deste estado, de permanecer um período em *busy wait* e outro período em estado inativo, simula em parte as ações de um usuário. Embora a simulação seja parcial, ela é suficiente e satisfatória para os nossos propósitos.

No entanto, ainda que seja simples efetuar esta alternância, é necessário administrar o tempo de permanência em atividade e inatividade. Este período de permanência em cada estado é regido por uma distribuição exponencial [Jai91], que fornece um valor pseudo-aleatório a cada chamada efetuada. Devido ao propósito do programa, não foi imposto rigor na teoria que fundamenta sua implementação, ainda que o método e processo científicos para simular tarefas correlatas [Fis92; JR94] sejam conhecidos.

A distribuição exponencial possui uma propriedade importante de não necessitar o momento do último evento ocorrido para inferir o instante do evento seguinte. Assim, a análise é facilitada, devido a esta "falta de memória" da distribuição. Sua função de densidade de probabilidade é:

$$P(x) = \begin{cases} \frac{1}{\theta} e^{-x/\theta}, & \text{se } x \geq 0 \\ 0, & \text{se } x < 0 \end{cases}$$

Na função acima, θ é o valor médio do tempo decorrido entre eventos sucessivos.

Para calcular o valor do intervalo entre eventos, é preciso obter a função inversa de $P(x)$. Para isso, utilizamos o método de transformação inversa.

Dados uma variável x e sua função de distribuição acumulada $F(x)$, a variável $u = F(x)$ possui distribuição uniforme no intervalo (0,1). Assim, gerando valores aleatórios uniformemente variados de u, é possível obter $x = F^{-1}(u)$. Ilustrando:

$$\begin{aligned} F(x) &= 1 - e^{-x/\theta} \\ u &= 1 - e^{-x/\theta} \\ 1 - u &= e^{-x/\theta} \\ x &= -\theta \ln(1 - u) \end{aligned}$$

Como u é uniformemente distribuída no intervalo (0,1), $1 - u$ também é uniformemente distribuída no mesmo intervalo. Uma forma simplificada, logo, seria:

$$x = -\theta \ln(u)$$

Esta distribuição é usada para modelar o intervalo entre eventos sucessivos, especialmente se os eventos são causados por um número grande de fatores independentes como, por exemplo, o intervalo entre requisições sucessivas para um sistema [Jai91].

Aplicado ao gerador de carga para consumo de processamento, o valor de θ é o intervalo médio entre cada iteração e também o intervalo de duração média de cada iteração. A partir disso, pretende-se simular a utilização da máquina por um usuário.

4.5.2 Automatização do processo

Com o intuito de automatizar o processo de execução dos experimentos, foi criado um programa em Perl especialmente dedicado a gerenciar os elementos envolvidos num experimento.

Todas as máquinas utilizadas para o experimento possuíam acesso remoto seguro. Especificadas as máquinas e os serviços que nelas devem ser iniciados, o sistema acessa-as quando executado, iniciando as tarefas solicitadas.

As tarefas que o sistema pode executar são:

- iniciar e interromper a execução de servidores de nomes;
- iniciar e interromper a execução de LRMs;
- iniciar e interromper a execução de GRMs;
- iniciar e interromper a execução de repositórios de componentes;
- interromper o experimento em andamento.

4.6 Análise, apresentação e interpretação dos dados

Neste item os dois sistemas, o de atualização de informações e o de localização de componentes, serão analisados segundo os dados coletados durante o funcionamento restrito às condições apresentadas nos itens anteriores. A apresentação dos dados coletados será feita por meio de gráficos representativos destes dados. A interpretação será realizada após aplicação de operações estatísticas, sobre as informações resultantes.

Algumas características do sistema mantêm-se válidas independente do experimentos em curso ou do subsistema analisado (disseminação de informações ou localização e execução):

- o tamanho máximo de cada notificação de atualização, incluindo os recursos, é variável, devido ao registro dos nós percorridos. Exceto essa particularidade, seu valor máximo é atingido quando todos os recursos registrados são atualizados. O tráfego gerado na rede não ultrapassa 300 bytes, levando em consideração todo o processo envolvido, o qual inclui a implementação do protocolo GIOP (IIOP) e TCP/IP. Estes valores foram obtidos com o auxílio da ferramenta Ethereal [Teaa];
- um único servidor de nomes foi utilizado para armazenar as referências de todos GRMs. Num ambiente real, onde a política administrativa de cada aglomerado deve ser respeitada ou a escalabilidade esteja comprometida, provavelmente uma configuração diferente seja necessária, como federação de servidor de nomes.

4.6.1 Sistema de disseminação de informações

Partindo-se dos dados obtidos pelo sistema de monitoramento de utilização de recursos, serão calculadas algumas estatísticas — média, desvio padrão, moda e intervalo de confiança — sobre a amostra e seguidas da interpretação destes resultados. Em todos os experimentos o valor mínimo encontrado de cada conjunto de observações numa máquina para o percentual de uso do processador foi de 0%.

O experimento foi conduzido de maneira a permanecer em execução por diversas horas. Em intervalos regulares de $10s$, o sistema de monitoramento coletou a utilização relativa do processador pelo GRM, além da memória consumida total até aquele momento. Eram mantidos contadores por parte de cada um dos GRMs representando o número de solicitações efetuadas pelos seus filhos, com registro do instante em que a requisição foi efetuada. O objetivo foi

corroborar nossas previsões de que um GRM exerce impacto significativamente baixo na máquina em que é executado.

A configuração do sistema utilizada nos dois experimentos está ilustrada na Figura 4.1. A máquina "grm2", entretanto, esteve ausente do experimento 2.

Experimento 1

Conforme descrito na Seção 4.2.8, neste experimento os períodos de checagem das condições dos recursos (p_1) e de atualização obrigatória (p_2) nos LRMs foram configurados para $5s$ e $10s$, respectivamente. Os GRMs estiveram isentos da checagem periódica do estado dos recursos (p_3) e foram configurados para atualizar seu pai obrigatoriamente ao término de um período de 5 minutos (p_4). A retirada da checagem periódica baseou-se na idéia de permitir que, primariamente, os eventos externos seriam os responsáveis pela emissão de uma notificação de mudança de estado dos recursos ao pai do GRM. Ou seja, a cada atualização do LRM no GRM, é disparada a verificação dos recursos.

A Tabela 4.1 apresenta as estatísticas calculadas conforme os resultados do experimento levando em consideração cada máquina onde se encontra um GRM em execução. As 5 estatísticas calculadas e nela presentes são baseadas numa amostra totalizando 3.500 observações, aproximadamente. Os valores foram obtidos numa freqüência de 6 observações por minuto, ou seja, a cada $10s$. As fórmulas usadas para o cálculo dos valores da tabela podem ser consultadas no Apêndice D. Como mostra a tabela, a média de utilização percentual relativa do processador pelo GRM é bem próxima a 0%, independente da máquina e da posição que ela se situa na árvore formada pela comunicação entre aglomerados do sistema (Figura 4.1). Os valores do desvio padrão apresentam uma baixa variabilidade dos dados coletados, o que sugere uma baixa variação da sobrecarga de processamento pelo GRM na máquina, representada pela média.

O grau de confiança utilizado foi de 0,95, indicando que, obtendo-se um número indefinido de amostras de mesmo tamanho a média da população estará em 95% dos intervalos constituídos pela média de cada amostra aplicando-se o intervalo apresentado na tabela de acordo com a máquina.

O valor máximo na tabela não representa o pico de utilização do processador, mas o valor máximo obtido por meio do programa coletor dos dados. No entanto, pode-se afirmar que num período de $10s$, o GRM não consome mais processamento do que esses valores apresentados, que são bastante baixos. Ademais, pelo fato dessas medidas serem baseadas em períodos fixos de $10s$, o percentual de utilização referente a todo o tempo em que a aplicação permaneceu em execução será próximo a 0%, resultado que podemos deduzir da relação entre o tempo efetivo de utilização e o tempo total.

Visualmente, os dados coletados sobre a utilização do processador em função do tempo estão caracterizados pelo gráfico de dispersão da Figura 4.5. Somente os dados de um GRM é apresentado, pois nos outros aglomerados todas as métricas verificadas são bastante próximas. É perceptível a concentração do percentual de uso do processador na abscissa nessas figuras, nunca ultrapassando 2%, o que demonstra visualmente o baixo consumo de processamento na máquina.

Neste experimento, as máquinas onde situavam-se os LRMs não sofriam qualquer intervenção artificial na utilização dos recursos. A utilização deu-se normalmente pelos próprios usuários. No entanto, a carga aplicada foi de baixíssimo impacto, influenciando minimamente o curso normal de execução do protocolo de disseminação de informações.

Houve um total aproximado de 3.400 mensagens enviadas pelo LRM da máquina `a202e08.escet.urjc.es` para o GRM de seu aglomerado notificando alguma alteração nos

Máquina	Média	Desvio padrão	Valor máximo	Moda	Intervalo de confiança
a202e02	0,01	0,05	0,58	0,00	$[0,008; 0,001]$
a202e08	0,05	0,19	1,75	0,00	$[0,045; 0,055]$
a202e14	0,05	0,16	1,36	0,00	$[0,046; 0,056]$
a202e20	0,05	0,15	1,36	0,00	$[0,042; 0,050]$
a202e26	0,05	0,16	1,17	0,00	$[0,044; 0,054]$
grm2	0,05	0,16	1,31	0,00	$[0,045; 0,055]$
villa	0,04	0,17	1,47	0,00	$[0,039; 0,048]$

Tabela 4.1: *porcentagem de consumo do processador pelo GRM (sem carga artificial)*

recursos. Considerando que o tempo total do experimento foi de aproximadamente 9,5 horas, a taxa de envio de mensagens foi, aproximadamente, de 0,1 mensagem por segundo.

Com as informações contidas neste experimento, é possível inferir a baixa interferência dos GRMs nas máquinas que os hospedam, quando levados para um ambiente real. Os dados obtidos referentes ao impacto dos LRMs em cada máquina [MK02] podem ser usados combinando-os com os apresentados nesta dissertação para um panorama abrangente do impacto do sistema nas máquinas.

Experimento 2

Este experimento difere do experimento 1 pela carga imposta nas máquinas, basicamente, com o intuito de elevar o consumo do processador artificialmente. Dessa forma, uma simulação fornece subsídios para inferirmos corretamente sobre o sistema em funcionamento num ambiente próximo ao real. Devido às próprias características dos sistema de disseminação de informações, a carga artificial induz atualizações mais freqüentes nas bases de dados que armazenam o estado dos recursos (GRMs). O programa descrito no Apêndice B é responsável por essas interferências e deve-se mantê-lo instalado em cada LRM do sistema. Como argumento, o programa recebe o valor de uma média, a qual define o intervalo em que o gerador de carga consome o recurso de processamento da máquina e o intervalo em que deve permanecer inativo. O valor desta média no experimento foi de $40s$.

Neste experimento manteve-se os períodos de $5s$ para a checagem das condições dos recursos (p_1) e $10s$ para a atualização obrigatória (p_2).

A Tabela 4.2, assim como a do Experimento 1 fornece estatísticas sobre a utilização do processador da máquina onde o GRM é executado. O total de observações neste experimento é de 3.000, aproximadamente. O grau de confiança utilizado nos cálculos foi 0,95. As máquinas `a202e02` a `a202e26` estão localizadas na Espanha, enquanto que a máquina `villa` pertence ao IME-USP.

É possível perceber na Figura 4.6 que, apesar dos valores serem mais distribuídos pelo gráfico, denotando um consumo mais freqüente da máquina, ainda assim, são bastante baixos, não ultrapassando 5% de uso do processador, como no experimento anterior. Destes resultados, podemos concluir que mesmo sob uma carga muito diferente da esperada num ambiente real, o sistema comporta-se de forma a não interferir significativamente no uso dos demais programas em execução.

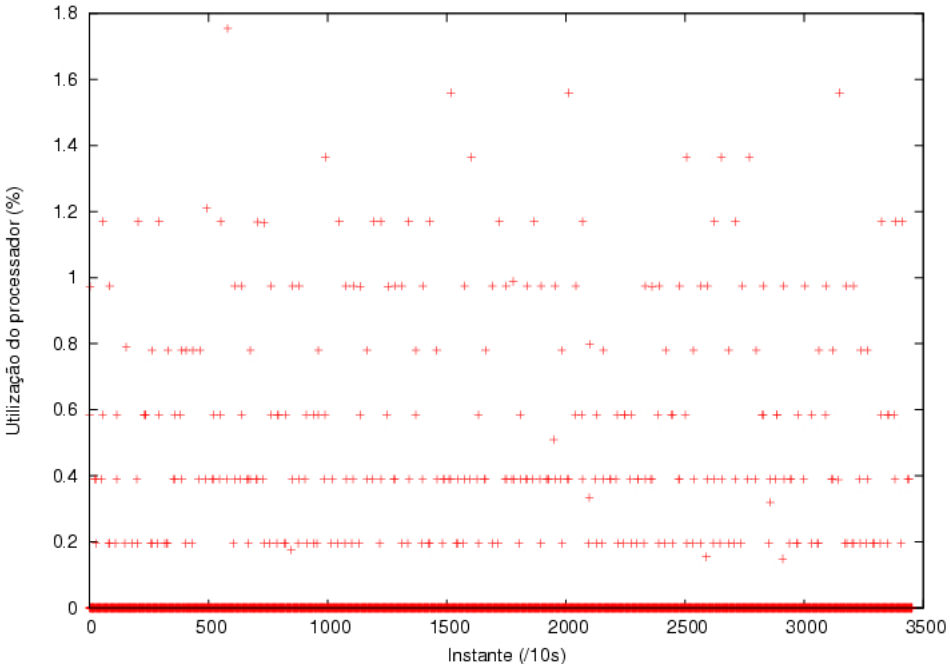

Figura 4.5: *Percentual de utilização do processador (máquina a202e08.escet.urjc.es)*

Comparativamente, como pode-se verificar pela Figura 4.7, os dados referentes à média de utilização do processador pelo GRM foi significativamente maior que os obtidos no experimento anterior, no qual as máquinas estavam isentas de carga artificial. No entanto, em relação ao uso do processador, ainda assim, pode-se afirmar que a utilização foi baixa.

Possivelmente, devido à resolução do relógio e da contagem de tiques de uso do processador, conseqüente da implementação do sistema operacional, combinados com o sistema de monitoramento, surgem conjuntos de dados bastante próximos, senão iguais, semelhantes a patamares, que surgem como linhas horizontais na Figura 4.6. Com o histograma da Figura 4.8 estes patamares, numericamente representados, ficam mais aparentes.

Neste experimento, mesmo sob uma situação anormal imposta pelo gerador carga, não se notou diferenças significativas em relação ao experimento anterior. O mecanismo usado para disseminação de informações mostrou-se apropriado para o ambiente em maior escala, segundo

Máquina	Média	Desvio padrão	Valor máximo	Moda	Intervalo de confiança
a202e02	0,45	0,28	1,56	0,00	$[0,430; 0,473]$
a202e08	2,26	0,69	4,87	2,14	$[2,243; 2,285]$
a202e14	1,59	0,28	3,12	1,56	$[1,582; 1,608]$
a202e20	1,50	0,43	3,12	1,36	$[1,440; 1,466]$
a202e26	1,62	0,75	3,70	1,36	$[1,610; 1,639]$
villa	0,58	0,31	1,72	0,00	$[0,557; 0,595]$

Tabela 4.2: *Porcentagem de consumo do processador pelo GRM (com carga artificial)*

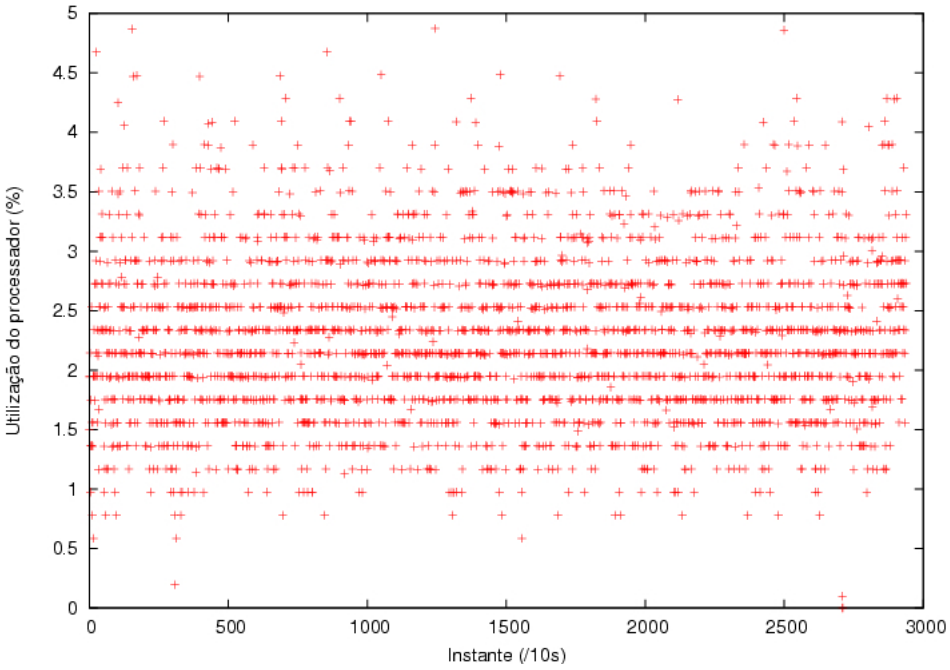

Figura 4.6: *Percentual de utilização do processador com carga artificial (máquina a202e08.escet.urjc.es)*

os dados obtidos, que não foram baixos como os da Figura 4.5, mas ainda assim, aceitáveis no uso cotidiano de uma máquina.

Experimento 3

Para demonstrar o baixo impacto do sistema de localização e execução de componentes, foi elaborado este experimento, com características semelhantes às dos dois experimentos anteriores, porém com uma diferença importante: a inclusão de clientes hipotéticos efetuando múltiplas requisições ao sistema, de acordo com uma distribuição exponencial com média $5s$. A duração do experimento foi de aproximadamente 9 horas.

O sistema de disseminação de informações permaneceu em funcionamento durante este experimento. Não seria interessante deixá-lo inoperante, pois a localização dos recursos depende fortemente das informações atualizadas. Quanto mais acurado estiver o sistema a respeito do estado dos recursos, mais próximas do valor desejado estarão as dicas (média e desvio padrão). Outra razão para mantê-lo em funcionamento foi a carga que ele gera no sistema. Ainda que exerça baixa influência no sistema, como mostrado nos experimentos anteriores, a intenção foi adicionar a carga que ele gera à do subsistema de localização. Desse modo, pode alcançar um nível mais próximo de um caso atípico que afetaria negativamente o sistema. Será mostrado, no entanto, que mesmo com a combinação dos dois protocolos, o impacto foi baixo.

A Tabela 4.3 demonstra o grau de utilização do processador numa máquina do experimento. O valor mais alto atingido foi 6,41% durante uma única medição — máquina "a202e33". Na mesma máquina, a média de utilização mostra que, em geral, o consumo permaneceu baixo, somente 1,15%. Podemos notar que a variação desta carga abrange um pequeno intervalo observando-se o desvio padrão de 0,58%. Provavelmente, estas porcentagens mantêm o mesmo nível de variabilidade, independente do componente, pois a tarefa de executá-lo ocorre

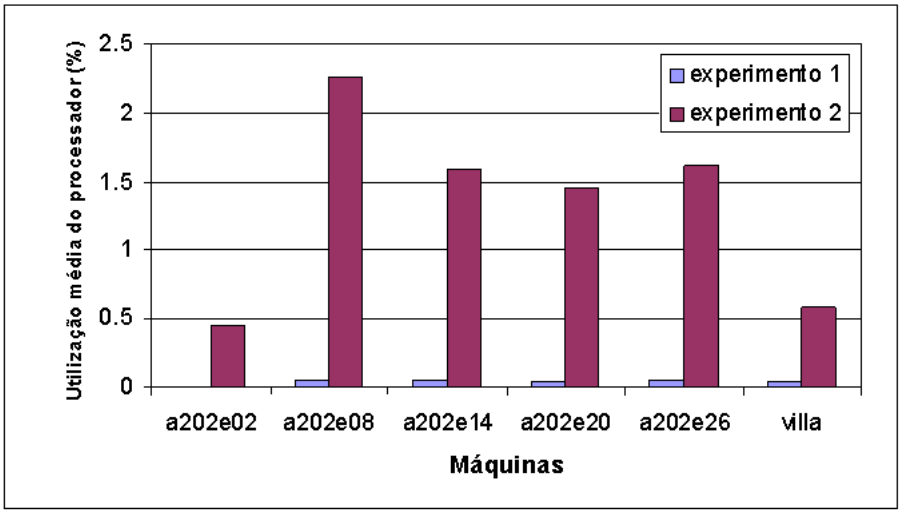

Figura 4.7: *Comparativo entre as médias dos experimentos para o sistema de disseminação de informações*

na máquina onde está situado o LRM. A Figura 4.9 ilustra a sobrecarga do GRM durante o experimento em outra máquina.

Algumas métricas referentes ao processo de requisição de componentes também foram inspecionadas. Elas estão presentes na Tabela 4.4. O número de requisições enviadas para os GRMs é dependente, em parte, do número de LRMs presentes no aglomerado em funcionamento. No entanto, como explicado anteriormente, podem ocorrer interrupções no funcionamento dos LRMs durante o experimento. O programa utilizado para realizar as solicitações foi configurado com uma média de $5s$ para gerar um número aleatório utilizado como período de espera entre cada nova solicitação. Supostamente, um usuário real deve ter um período de latência maior entre cada requisição. A sobrecarga na rede durante o período foi pequeno, chegando a 0,21

Máquina	Média	Desvio padrão	Valor máximo	Moda	Intervalo de confiança
a202e02	0,37	0,60	1,76	0,20	$[0,36; 0,38]$
a202e08	1,31	0,67	5,07	0,98	$[1,29; 1,33]$
a202e14	0,91	0,48	3,90	0,78	$[0,90; 0,93]$
a202e19	0,13	0,12	1,17	0,00	$[0,12; 0,13]$
a202e20	1,01	0,36	2,72	0,98	$[1,00; 1,03]$
a202e26	0,83	0,38	2,53	0,78	$[0,81; 0,84]$
a202e33	1,15	0,58	6,41	0,78	$[1,12; 1,18]$

Tabela 4.3: *Estatísticas para o experimento de localização e execução de componentes*

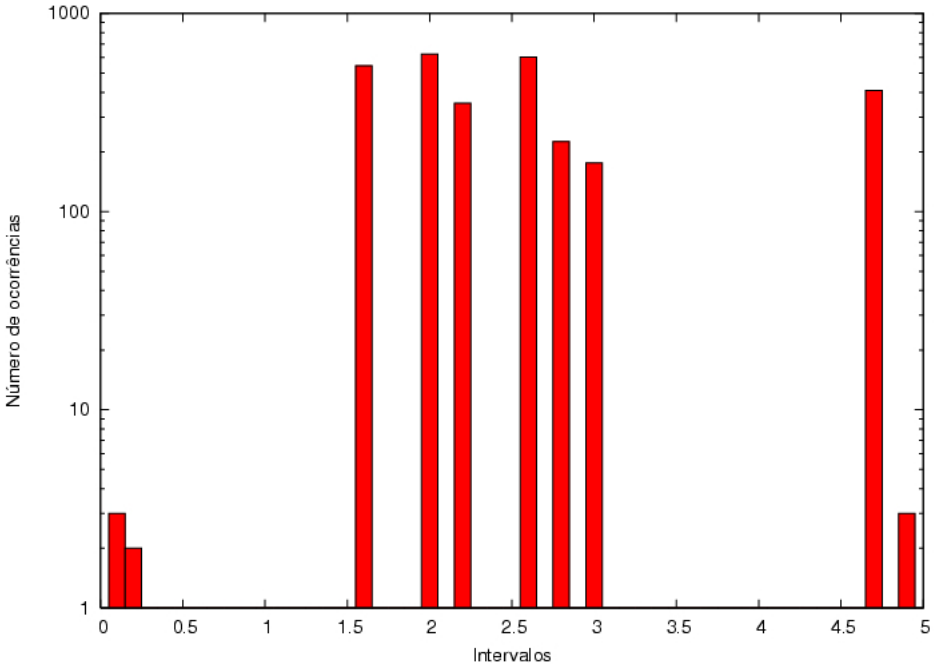

Figura 4.8: *Histograma de utilização do processador com carga artificial (máquina a202e08.escet.urjc.es)*

mensagens/s, no máximo. Conseqüentemente, pelo fato de que o processo de envio de uma mensagem aproxima-se de 2 KB, a quantidade de informação inserida na rede é bastante baixa. O tempo médio para atendimento refere-se ao aglomerado por inteiro, ou seja, é a média dos valores obtidos por todas as máquinas que continham um LRM no aglomerado e, portanto, um cliente efetuando requisições.

Na Tabela 4.4, os nomes das máquinas representam os aglomerados (coluna "Aglomerados"). A coluna "Requisições atendidas" indica o número de mensagens com origem no aglomerado que foram atendidas, isto é, que foi encontrado um local adequado para executar o componente segundo os critérios presentes na requisição. O inverso ocorre na coluna "Requisições não atendidas", onde são contabilizadas as mensagens que não puderam ser executadas em nenhuma parte do sistema, em qualquer aglomerado. A quantidade mais alta de mensagens que não puderam ser atendidas era esperado, em vista da restrição presente na mensagem, que diz respeito à quantidade de memória volátil disponível. Por exemplo, o programa gerador de requisições pode ter criado o intervalo `RAM_total >= 90 and RAM_total < 100`, o qual não pôde ser atendido. O "Total de mensagens" é a soma dos valores localizados nas colunas de requisições atendidas e não atendidas, respectivo ao aglomerado. A coluna "Taxa de recebimento" mostra o número de mensagens inseridas na rede a cada segundo intra-aglomerado e inter-aglomerado. Por fim, na última coluna, apresenta-se o tempo médio de atendimento de uma requisição para o conjunto de clientes (programas geradores) num aglomerado. Podemos observar que o tempo é relativamente baixo e, aparentemente, com a altura da árvore usada no experimento, não é possível inferir o quanto a altura da árvore influencia no intervalo de espera para que uma requisição seja respondida.

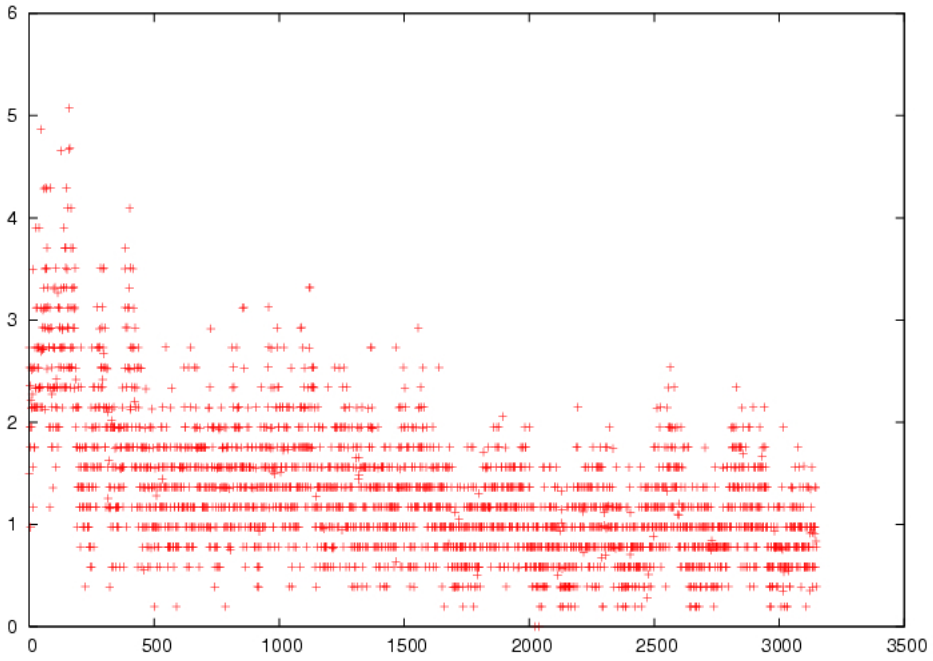

Figura 4.9: *Percentual de utilização do processador (máquina a202e08.escet.urjc.es)*

4.7 Conclusões dos Experimentos

As informações colhidas durante os experimentos permitiram inferir sobre o impacto do sistema e sua escalabilidade. Nos três experimentos, foi possível verificar a baixa utilização do processador pelo GRM, ilustrados nas tabelas e nos gráficos, A média de utilização percentual relativa do processador foi bastante baixa, atingindo valor máximo de 6,41% no terceiro experimento, entre todos os experimentos. Além disso, o desvio-padrão indicou baixa dispersão nos dados coletados. É importante salientar que a coleta de dados era efetuada a cada período de 10s e, dessa forma, o percentual de utilização relativo ao tempo total que um GRM permaneceu em execução aproxima-se de 0%.

O primeiro e o segundo experimento abordaram o Sistema de Disseminação de Informações.

Aglomerado	Requisições atendidas	Requisições não atendidas	Taxa de recebimento (mensagens/s)	Tempo médio para atendimento (s)	Desvio padrão	Tempo máximo para atendimento (s)
a202e02	190	1497	0,15	0,20	0,03	0,29
a202e08	688	4169	0,21	0,22	0,03	0,37
a202e14	811	5868	0,15	0,25	0,04	0,43
a202e19	30	190	0,01	0,18	0,03	0,24
a202e20	363	1970	0,07	0,20	0,03	0,32
a202e26	166	1286	0,21	0,21	0,03	0,32
a202e33	388	6509	0,21	0,22	0,05	0,57

Tabela 4.4: *Solicitações recebidas nos aglomerados*

Eles se diferenciaram, essencialmente, pela imposição de uma carga artificial praticada no segundo experimento e ausento no primeiro. Mesmo com esta sobrecarga artifical, não se notou diferenças significativas comparando-se os resultados dos dois experimentos.

No terceiro experimento, referente ao Sistema de Localizaçao e Execução de Componentes, também pôde-se constatar o baixo impacto do sistema nas máquinas, assim como nos outros dois experimentos. Além disso, foram observados os tempos de atendimento a requisições feitas artificalmente por uma aplicação-cliente. Os valores médios para atendimento de requisições foram proximos a $0{,}20s$, com valor máximo de $0{,}57s$.

De forma geral, podemos constatar pelos experimentos que os GRMs interferiram de maneira aceitável nas máquinas onde estavam sendo executados, o que nos leva a inferir que o sistema pode receber mais aglomerados participantes sem detrimento do serviço oferecido.

Capítulo 5

Trabalhos Relacionados

A seguir são descritos dois dos sistemas de grade mais conhecidos e a abordagem de cada um para o problema de gerenciamento de recursos em sistemas distribuídos baseados em suas características e finalidades.

Vários grupos de pesquisa têm se dedicado ao desenvolvimento de sistemas distribuídos que oferecem suporte a gerenciamento dinâmico de recursos. Alguns seguem rumos ligeiramente diferentes, enquanto outros seguem a mesma direção que a do projeto aqui proposto. Neste capítulo, apresento os projetos relacionados, destacando os aspectos semelhantes e divergentes, positivos e negativos correspondentes a este trabalho.

Legion e Globus são dois grandes projetos que oferecem suporte à arquitetura de grade similarmente e, em alguns aspectos, complementam-se. Eles surgiram em meados de 1993 e 1996, respectivamente, e são pioneiros neste campo da computação. O Condor-G e o Nimrod são projetos desenvolvidos para solucionar o problema de gerenciamento de recursos ociosos; ambos podem ser utilizados pelo Globus.

Grandes avanços foram alcançados por meio das pesquisas dedicadas a esses ambientes, particularmente em gerenciamento de recursos distribuídos, descoberta de recursos e seus sistemas de informação, os quais representam e armazenam dados sobre as condições dos recursos e suas descrições.

5.1 Globus

O Globus [FK97], que surgiu em 1996, tem como objetivo oferecer uma infra-estrutura baseada nos conceitos de grades computacionais para aplicações relacionadas às áreas de engenharia e ciência em geral, a fim de integrar recursos computacionais heterogêneos possivelmente localizados em diferentes áreas geográficas num único sistema lógico.

Seus componentes fazem parte de um conjunto de serviços de baixo nível: o *Globus Toolkit*. Eles podem ser usados simultânea ou independentemente para desenvolver componentes de nível mais alto, como aplicações e ferramentas. Orientando-se nos serviços de baixo nível, ele diferencia-se do caminho adotado no 2K, onde os serviços de alto nível implementados pelo *middleware* representam abstrativamente os recursos (ou serviços) de baixo nível.

Os componentes do *toolkit* possuem funcionalidades, políticas e mecanismos básicos que permitem, especialmente, monitorar, criar, gerenciar e alocar processos; encontrar serviços; autenticação de serviços; detectar falhas em componentes do sistema ou em aplicações.

Para gerenciar e manter todo o conjunto de recursos disponíveis, assim como os dados que representam seus estados, o Globus dispõe de dois serviços básicos:

- *Globus Resource Allocation Manager* (GRAM[CFK+98]): oferece suporte para alocação de recursos, monitoramento e gerenciamento de recursos.

- *Monitoring and Discovery Service* (MDS[CFFK01]): é um serviço baseado em componentes distribuídos que oferece meios para acesso a informações sobre o estado do sistema e instrumentos para possíveis reconfigurações. Este serviço também incorpora mecanismos para descobertas de serviços e disseminação de informações a respeito do estado do sistema. Funcionalmente, ele agrega informações sobre os recursos oferecidos no sistema, assim como seus estados, por meio de dois subsistemas: *Grid Resource Information Service* (GRIS) e *Index Information Service* (GIIS).

No modelo de gerenciamento de recursos do Globus, uma aplicação interage diretamente com três componentes, cada um responsável por um segmento no processo de alocação de recursos. São eles: *resource brokers*, que lidam com especificações de alto nível baseadas numa linguagem desenvolvida para esta finalidade, a RSL, traduzindo o pedido para algo mais concreto, de acordo com o gerenciador de recursos local; o *co-allocator*, cuja função é coordenar a alocação e o gerenciamento dos recursos em diversos domínios e, por fim, o serviço de informações.

Informações sobre a disponibilidade e características dos recursos são responsabilidades do serviço de informações. Estas informações são usadas para localizar recursos com determinadas características, encontrar o gerenciador de recursos associado a um determinado recurso, assim como suas propriedades. No Globus, o sistema de informações utilizado é o MDS [CFFK01] (*Monitoring and Discovery Service*). No MDS, a representação dos dados e uma interface para programação são baseadas no LDAP [YHK95] (*Lightweight Directory Access Protocol*). Dessa forma, o serviço local associado com o MDS é também um servidor LDAP, que mantém o servidor informado sobre o estado dos recursos do domínio. O MDS encarregado de todo o sistema possui uma visão global de todos estes servidores.

Percebe-se, neste ponto, que o MDS e o GRAM possuem seus paralelos no Sistema de Gerenciamento de Recursos do 2K. O LRM corresponde em parte ao MDS e o GRM pode ser inserido integralmente no conceito do MDS somado ao LDAP, devido às características análogas entre os sistemas, como a disseminação de informações, exposta na Seção 3.2. O GRAM, por sua vez, sendo responsável pela alocação e reserva dos recursos, mantém relação mais estreita com o LRM e com DSRT, do que com o GRM, que envolve-se na localização dos recursos solicitados pelo cliente.

Como citado anteriormente, o gerenciador de recursos local, a entidade de nível mais baixo da arquitetura, é o GRAM, cujas responsabilidades são:

- processar a RSL representando recursos requisitados por algum outro participante, rejeitando a requisição ou iniciando um ou mais processos que satisfaçam o pedido;

- habilitar monitoramento e gerenciamento dos processos criados;

- atualizar periodicamente o serviço de informações com o estado dos recursos num determinado instante, além da disponibilidade de cada recurso.

Um GRAM serve como meio de ligação entre um ambiente de grade de grande área e entidades capazes de criar processos e que disponibilizem recursos computacionais. No entanto, um gerenciador de recursos não está restrito a uma máquina em particular, mas sim a um serviço que atua como intermediário para acesso a um ou mais recursos.

Uma requisição encaminhada para um GRAM deve ser suficientemente concreta na medida em que o GRAM seja capaz de identificar os recursos locais que atendem à requisição,

sem que seja necessário qualquer interação com quem a gerou. Uma implementação de um GRAM de acordo com este modelo pode, ele mesmo, agendar os recursos ou ainda mapear a requisição para algum mecanismo local de gerenciamento de recursos, como o Condor [LLM88], o LoadLeveler [IBM93] ou o LSF [Zho92].

Ao executar um processo, um *job handle* é devolvido para o cliente que efetivou a solicitação pelo *jobmanager*. Entre outras utilidades, ele pode ser usado para monitorar o progresso da tarefa que está em execução e pode ser repassado para outros processos. Este recurso permite que diversas estratégias distintas de escalonamento sejam criadas. A escalabilidade no Globus é dependente do serviço executado pelo *jobmanager*.

Outros dois módulos constituintes possuem notada importância no sistema:

- *Global Access to Secondary Storage* (GASS) [BFK+99]: é um serviço que define um espaço de nomes global por meio de URLs e permite que aplicações acessem arquivos remotos por meio de interfaces de entrada e saída (E/S);
- *Grid Security Infrastructure* (GSI) [FKTT98]: oferece comunicação e autenticação seguras por uma rede possivelmente desprotegida. Ele é baseado em criptografia de chave-pública, certificados de acordo com o protocolo X.509 e soquetes seguros (SSL). Dessa forma, ele suporta comunicação segura entre elementos da grade computacional; oferece gerenciamento distribuído da segurança e assinatura digital única para os usuários.

5.2 Condor-G

O Condor-G [FTL+01] é um sistema desenvolvido na Universisdade de Wisconsin com o objetivo de incorporar políticas, modelos e mecanismos de gerenciamento de recursos a grades computacionais, especificamente, no Globus. O sistema, que é um arcabouço, segue preceitos básicos de ambientes de grade, como descoberta dinâmica de recursos, transparência de localização dos recursos computacionais por parte dos usuários e acompanhamento do estado dos processos criados.

O Condor-G beneficia-se de avanços alcançados por outro projeto, o Condor [LLM88; BL00], no campo de sistemas para gerenciamento de recursos.

Essencialmente, o Condor-G baseia-se nos protocolos do *Globus Toolkit* e nas funcionalidades de gerenciamento de recursos para um único aglomerado oferecidas pelo Condor. Combinados, estes dois sistemas permitem ao usuário utilizar recursos disponíveis além de seu domínio gerencial, isto é, pertencentes a outro domínio. Este processo é feito de forma que, na perspectiva do usuário, os recursos estão disponíveis no seu próprio domínio. Antes de aprofundar em detalhes do Condor-G, segue um introdução de algumas características do Condor, seu sistema base.

O sistema Condor restringe-se a gerenciar os recursos computacionais de um único conjunto de máquinas conectadas por meio de uma rede local, tornando-o um sistema uniforme. Ele contorna alguns problemas comuns que surgem em outros sistemas similares decorrentes da abordagem no desenvolvimento do algoritmo de escalonamento. Nestes algoritmos, assume-se que os recursos estarão disponíveis indefinidamente para efeito de cálculo dos escalonamentos. A maior parte dos algoritmos existentes são direcionadas para um conjunto restrito de tipos de aplicações, ou seja, se um usuário possuir duas aplicações, uma que necessite de execução em série e outra que execute paralelamente, ele terá de enviá-las para sistemas diferentes, de acordo com a finalidade. Como conseqüência da adoção desta abordagem, outros problemas surgem:

incremento na dificuldade de administração e particionamento do aglomerado de acordo com seu tipo de recurso.

A estratégia do Condor para contornar estes obstáculos é combinar perspectivas de escalonamento dedicado, descrito acima, com escalonamento oportunista, que executa processos em recursos que supostamente não estarão disponíveis por todo o ciclo de vida do processo. Baseado nisso, o sistema interrompe o processo num determinado instante pré-estabelecido e grava seu estado num arquivo. Este arquivo é usado posteriormente para restabelecer o processo no ponto exato onde foi suspenso.

Um ponto que pode trazer problemas dependendo da aplicação específica em que ele é empregado é o modo de organização do sistema, que não é hierárquico.

Mecanismos para descoberta de recursos e escalonamento, no entanto, ainda não foram completamente explorados para ser usados no Condor-G. Uma idéia cogitada é construir um mediador recursos que seria parte integrante do agente descrito acima e que combina informações sobre autorização de usuários, requisitos da aplicação e estado dos recursos, obtidos pelo MDS, para então construir uma lista dos candidatos. O estado dos recursos candidatos é consultado e a aplicação é encaminhada para o mais apropriado segundo o resultado da consulta. Além disso, o usuário pode discriminar detalhadamente seu critério de preferência para a lista obtida, da mesma forma que é possível no modelo de gerenciamento de recursos do 2K e do trabalho exposto nesta dissertação.

5.3 Legion

O Legion [CKKG99; GW+97; GFKH99; NHG01] é um projeto desenvolvido na Universidade de Virginia que consiste numa infra-estrutura para grades computacionais. Ele é um sistema operacional de grades orientado a objetos, assim como o 2K, implementado como um middleware para permitir interação entre recursos computacionais possivelmente heterogêneos.

Similarmente a outros ambientes de grades, o Legion oferece acesso transparente aos recursos disponibilizados por meio de uma visão unificada dos elementos computacionais que são incorporados ao sistema.

No ambiente do Legion, os objetos que o compõem representam os elementos da grade. Eles são controlados por outros objetos responsáveis por criá-los, organizar os instantes para execução, ativá-los ou desativá-los. Estes objetos podem ter seus estados gravados para reiniciação posterior e, além disso, podem ser reativados se algum outro objeto comunicar-se com eles.

De acordo com suas funções, esses objetos principais são divididos em 3 grupos. Objetos `Class` no Legion possuem duas funções. Ainda que o nome induza a algo genérico, no Legion, `Classes` são também entidades ativas e agem como gerenciadores de suas instâncias. Os dois objetos restantes representam os tipos básicos de recursos: `Hosts` e `Vaults`. Cada um possui uma classe guardiã. Objetos `Host` encapsulam características da máquina, como processador e memória; são responsáveis por instanciar objetos no processador.

Como forma de permitir maior escalabilidade, `Hosts` permitem reservas para uso futuro sendo que a forma de como a reserva é feita depende de sua implementação. A implementação das reservas codifica o `Host` e `Vault` que serão usados para execução do objeto. `Vaults` são abstrações de recursos que representam armazenamento persistente no Legion. Eles somente participam dos estágios iniciais do processo de escalonamento, verificando se eles são compatíveis com o ambiente onde estão. `Hosts` também possuem um mecanismo para definir disparo de eventos.

Isto permite que uma máquina inicie a migração de um objeto quando sua carga alcançar um determinado limite.

O modelo de gerenciamento de recursos suporta escalonadores definidos pelo usuário para interagir com a infra-estrutura. Os componentes do modelo são os recursos básicos (`Hosts` e `Vaults`), a base de dados com as informações do sistema (`Collection`), a implementação do escalonador (`Enactor`) e um monitor de execuções feitas. O `Collection` corresponde ao GRM no 2K, que possui um repositório de dados sobre o estado dos recursos. Os `Hosts`, `Vaults` e o `Enactors` apresentam funções exercidas pelo LRM. Estes dois componentes do 2K são descritos na Seção 2.1.2. Os objetos `Host` possuem na sua interface três grupos de funções: gerenciamento de reserva, gerenciamento de objeto e divulgação de informação.

Comparativamente ao Globus, descrito no item anterior, o Legion assemelha-se em alguns pontos e difere noutros. Eles compartilham os objetivos técnicos, a característica dos ambientes onde são implantados e gerenciam e são destinados a um grupo com perfil de usuário muito similares. Porém, como já pôde ser identificado até o momento, a concepção do arquitetura de cada um são nitidamente distintos. O Legion propõe uma diminuição da complexidade para o usuário e oferece ao programador uma visão homogênea, orientada a objetos, dos recursos disponíveis sob seus cuidados. Já o Globus, constitui-se de um agregado de serviços essenciais, cada um com responsabilidades bem definidas conforme sua atuação no sistema.

5.4 GriT

Seguindo a proposta de *middleware*, o GriT (*Grid TAO*) [GN02], um ambiente de grade computacional, baseando-se no *CORBA Component Model* (CCM)[1] [Gro00a], propõe-se a aprimorar o Component Integrated ACE ORB (CIAO), uma implementação do CCM.

O GriT enquadra-se no conceito de provedor de serviços em grades computacionais (*Grid Service Provider*), retirando da aplicação a necessidade de obter informações sobre localização e características dos recursos oferecidos.

Com maior importância entre as partes constituintes do GriT, encontra-se o *Meta Resource Broker* (MRB), cujos atributos funcionais compreendem mediar as requisições efetuadas pelas aplicações e, como resultado, fornecer os recursos com garantia da qualidade de serviço solicitada. A tarefa de procurar recursos é delegada a outros objetos. O MRB mantém uma visão abstrata dos recursos — banco de dados ou computadores, por exemplo — denominados de recursos "virtuais", no sentido em que ele não mantêm quaisquer recursos físicos, mas somente uma abstração destes, os quais podem pertencer a diferentes domínios de posse dos recursos concretos. De forma análoga, o GRM e o LRM garantem as mesmas funcionalidades.

O mecanismo de busca pelos objetos consiste em particionar a requisição, possivelmente uma combinação de vários recursos, de forma tal que cada objeto, individualmente, fique encarregado de localizar um fragmento da requisição. Isso é executado analogamente a uma transação: se um deles não obtiver êxito na busca do recurso, todos os outros que conseguiram retornam ao seu estado inicial do processo, ou seja, ocorre um *rollback*. Assim, o mecanismo permite que todo o processo seja efetuado paralelamente, sendo bastante vantajoso em relação ao tempo total gasto para atender a requisição integralmente e também favorece a escalabilidade. Neste ponto, há uma vantagem sobre o sistema que foi foco nesta dissertação: o paralalelismo na busca dos recursos viabilizado pelo particionamento da requisição. O GRM e o LRM não oferecem esta funcionalidade; a requisição é levada nó a nó integralmente.

[1] Uma breve introdução ao CCM pode ser encontrada em: http://openccm.objectweb.org/doc/ccm.html

O CCM entra no desenvolvimento dos objetivos individuais descritos acima. Os componentes encapsulam a lógica e as políticas para gerenciar os recursos "virtuais". Dessa maneira, esses objetos atuam intermediariamente entre os recursos solicitados e a aplicação.

Capítulo 6

Conclusão

O intuito do trabalho foi implementar um meio de interligar aglomerados de modo que a sobrecarga gerada nas máquinas quando em funcionamento pleno, fosse aceitável, além de permitir a inclusão de outros participantes de forma a não deteriorar sobremaneira sua execução. Era importante cumprir este requisito, pois os participantes do sistema devem ter um papel secundário nos computadores que coabitam com outros processos. Eles não devem intervir de maneira negativa a ponto de ser significativo para o usuário. Ademais, a escalabilidade foi satisfatória, segundo os valores obtidos nos experimentos. Nesse sentido, como pôde-se observar nos resultados apresentados nas seções anteriores, a implementação garantiu os requisitos inicialmente propostos.

Certamente, o sistema possui pontos ainda a serem investigados, como novas funcionalidades e abordagens diferentes para a mesma arquitetura. Estes pontos são comentados nas seções a seguir.

O trabalho apresentado nesta dissertação levou a 2 publicações:

- Gerenciamento de Recursos Distribuídos em Sistemas de Grande Escala. Jeferson Roberto Marques e Fabio Kon [MK02]. Publicado no XX Simpósio Brasileiro de Redes de Computadores em maio de 2002.
- Design, Implementation, and Performance of an Automatic Configuration Service for Distributed Component Systems [KMY+05]. Fabio Kon, Jeferson Roberto Marques, Tomonori Yamane, Roy H. Campbell, and M. Dennis Mickunas. Aceito para publicação no *journal* "Software: Practice and Experience", 35(7). Será publicado em maio de 2005.

6.1 Trabalhos futuros (complementações)

Nesta seção são expostas idéias que surgiram durante o desenvolvimento deste sistema, as quais podem tornar o sistema mais robusto e seguro, além de diversificar suas funcionalidades. No entanto, tais idéias estavam fora do escopo que compreende o projeto. Assim, seguem algumas discussões relativamente superficiais referente a essas características desejáveis para enriquecimento do sistema.

6.1.1 Alternativa de comunicação inter-aglomerados

Na concepção e implementação do sistema descrito nesta dissertação, a interação final entre usuários é manipulada efetivamente pelos LRMs, isto é, os LRMs interagem entre si diretamente, estabelecendo uma conexão entre o cliente e o outro participante, que faz o papel de

servidor. Uma proposta diferente para a comunicação entre os aglomerados seria utilizar outro componente do sistema para ser responsável pelas comunicações entre máquinas de outros aglomerados, por exemplo, com os LRMs efetuando a localização diretamente. O objetivo dessa abordagem é manter as máquinas sob controle mais restrito, eliminando preocupações adicionais com segurança para acesso às máquinas do aglomerado. Outra vantagem, particularmente importante, é a utilização de redes lógicas com IPs destinados a redes privadas. Esses IPs não são acessíveis pela Internet propriamente dita. Dessa maneira, um participante faria o papel de *proxy*, isto é, um intermediário, praticando as relações entre LRMs de um determinado aglomerado com de outros, transparentemente.

Poderia ser desenvolvido um componente adicional aos já existentes nesse sistema. Ele estaria incumbido de encaminhar, mapear e organizar o tráfego de requisições e comunicações em geral entre as redes. Usualmente dá-se o nome de *gateway* a uma entidade definida por essas características.

Poderia ainda, utilizar-se do GRM para tal encargo, pois ele age numa posição naturalmente estratégica e facilitadora de inter-relacionamentos, além de possuir, necessariamente, um IP válido, acessível pela Internet. Aproveitando-se de sua posição sobre as delimitações da rede, é possível incrementá-lo com a funcionalidade extra de *proxy*.

6.1.2 Segurança

Os pontos do sistema que criam possibilidades para que aplicações usufruam ilicitamente são visíveis. Certamente, preencher estas lacunas de vulnerabilidades é necessário para que o sistema seja implantado e execute de maneira confiável suas tarefas, prevenindo que um usuário ou aplicação comprometa o uso do conjunto computacional. Os problemas descritos neste item compreendem o conjunto mais comum em sistemas distribuídos: comunicação segura, lista de controle de acesso e autenticação dos participantes do sistema.

A maioria dos sistemas operacionais não isolam criteriosamente as aplicações entre si. Por esta razão, uma aplicação, potencialmente, pode prejudicar o funcionamento do sistema por completo. Um modo de contornar esses problemas é incorporar, às aplicações, meios de identificarem-se ao interagirem entre si.

Uma maneira comum de interferir na privacidade dos usuários é a grande margem de risco que existe nos dispositivos utilizados na Internet, permitindo interceptar dados transmitidos entre participantes do sistema. A solução para isto é relativamente simples e bastante difundida em diversos ambientes computacionais: os canais seguros (criptografados), cujo protocolo comumente utilizado é o SSL [HE95] e, mais recentemente . Implantado esse protocolo, dois participantes podem fazer uso da rede sem a preocupação de que dados privados venham a ser interpretados, caso sejam interceptados. Pelo fato de ser necessário incorporar o mecanismo somente nos participantes do sistema, não há necessidade de estender para toda a rede sob qual ele funciona, ou seja, esta solução permite que o sistema continue a ser executado numa rede insegura.

É aceitável que os recursos presentes nos aglomerados não devam ser disponibilizados em sua totalidade para os usuários, ou ainda, queira manter-se um controle administrativo do aglomerado, por exemplo, controlar quais tipos de recursos e para quem estão autorizados. Para isso, uma solução é a inclusão no sistema do conceito de lista de controle de acesso. Por meio de um mecanismo similar a um serviço de nomes, cada recurso teria uma relação daqueles que o podem utilizar e, se for aplicável, uma especificação mais detalhada, por exemplo, por quanto tempo pode utilizar. E para que se esteja seguro de que determinado usuário é realmente quem

ele se identificou como tal, é desejado que haja um modo confiável de identificação, como os certificados seguros.

6.1.3 Inclusão de recursos

Atualmente, os recursos são definidos estaticamente. Esta característica restringe a oferta de recursos aos usuários somente àqueles previamente especificadas e compartilhadas com todos os participantes, embora facilite muito a administração e controle. É bastante atraente a idéia de agregar mais tipos de recursos ao sistema de forma mais dinâmica. As duas potenciais soluções aqui propostas demandam uma alteração na concepção do *design* do sistema, precisamente em relação ao LRM e ao GRM.

Uma refatoração do LRM, retirando-lhe a responsabilidade de averiguar periodicamente a disponibilidade dos recursos seria adequada e traria maior flexibilidade, garantindo maior coerência na distribuição de responsabilidades. Um novo objeto ficaria responsável por esta tarefa, independentemente do LRM. Estendendo ainda mais o conceito, um subsistema estaria exclusivamente encarregado de gerenciar as alterações dinâmicas dos recursos na máquina. Ainda mais importante seria a criação de uma entidade encarregada de receber as informações atualizadas provenientes de cada uma das máquinas — no momento esta tarefa é designada ao GRM, de forma que esta responsabilidade também seria dele retirada.

Um cenário hipotético que ilustra um benefício desta abordagem é a adição de um novo domínio ao sistema, trazendo consigo novos tipos de recursos computacionais para serem compartilhados. O aparecimento deste novo domínio seria percebido dinamicamente pelos usuários, sem que nenhum programa seja reiniciado.

Por meio desta nova concepção, os recursos seriam descritos através de um modelo de dados padronizado, por meio de texto contidos em arquivos presentes em cada participante. Assim, atinge-se maior controle e dinamismo para todo o sistema. No modo atual, é necessário que seja gasto tempo extra para compilar novamente os objetos, caso alguma alteração deva ser efetuada na IDL em relação à descrição dos recursos disponíveis.

6.1.4 Diagnóstico e gerenciamento de rede

Uma funcionalidade não coberta no projeto é a administração e disponibilização de meios de controle de carga da rede e informação sobre seu uso. É esperado que uma aplicação tenha restrições quanto à sua execução em relação à vazão na rede, por exemplo. No entanto, não há suporte para garantia de recursos relacionados à rede no sistema apresentado nesta dissertação. O projeto que desenvolveu o DSRT, utilizado pelo LRM, possui planos para incluir reserva de recursos de rede, do qual o LRM poderia obter meios para oferecer esta funcionalidade às aplicações.

6.1.5 Tolerância a falhas

Ocasionalmente, um sistema pode ser afetado negativamente por algum evento indesejado no ambiente em que se encontra, comprometendo a execução esperada de suas funções. Por esta razão, torna-se desejável a construção de um conjunto de medidas para impedir ou amenizar o mau-funcionamento decorrente, inclusive formalmente definidos acerca das dependências entre as partes constituintes do sistema [Pow92]. Há situações em que são necessários sistemas de tolerância a falhas complexos para manutenção das tarefas executadas pelo sistema, pois podem ser afetados a ponto de impedir completamente suas atividades. E, à medida que o sistema descentraliza-se, objetivando dinamismo e escalabilidade, a tolerância a falhas torna-se mais crítica. O CORBA

possui, em sua especificação, meios de lidar com essas circunstâncias. O TAO, implementação do CORBA, implementa as políticas e mecanismos previstos no CORBA contornar essas eventualidades. Ele implementa o padrão FT-CORBA (*Fault-Tolerant CORBA*) [Gro00b], destinado ao gerenciamento de falhas.

No projeto a que se propõe esta dissertação, algumas situações suscetíveis a falhas podem ser corrigidas com auxílio do padrão FT-CORBA de acordo com os seguintes pontos:

- *persistência:* as informações dos recursos armazenadas no *trader* podem ser mantidas em disco para uso posterior, caso ele seja interrompido em circunstâncias não-usuais.
- *replicação de objetos:* os recursos disponíveis são naturalmente replicados em cada dentro de cada aglomerado e analogamente entre aglomerados;

Nesta dissertação apresentamos o 2K, um sistema operacional que suporta interação entre componentes presentes em arquiteturas heterogêneas que, inicialmente, operava dentro dos limites de um aglomerado. O projeto exposto demonstrou a expansão destes limites através do protocolo inter-aglomerado, o impacto da nova configuração no ambiente de execução e sua escalabilidade. Futuramente, esperamos torná-lo mais completo em diferentes aspectos, por exemplo, melhorando a tolerância a falhas e a segurança no uso dos recursos disponíveis, ambos expostos nesta seção.

Referências Bibliográficas

[BBC+04] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, , and Mike Wawrzoniak. Operating System Support for Planetary-Scale Services. In *Proceedings of the First Symposium on Network Systems Design and Implementation (NSDI)*, March 2004.

[BFK+99] Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, and Steven Tuecke. GASS: A Data Movement and Access Service for Wide Area Computing Systems. In *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pages 78–88, Atlanta, GA, 1999. ACM Press.

[BL00] J. Basney and M. Livny. Managing network resources in Condor. In *High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium*, pages 298–299. IEEE Computer Society Press, August 2000.

[bog] BogoMips mini-Howto. http://www.tldp.org/HOWTO/BogoMips/.

[CC91] Roger S. Chin and Samuel T. Chanson. Distributed, object-based programming systems. *ACM Computing Surveys*, 23(1):91–124, 1991.

[CFFK01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, page 181. IEEE Computer Society, 2001.

[CFK+98] Karl Czajkowski, Ian T. Foster, Nicholas T. Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82. Springer-Verlag, 1998.

[CKKG99] Steve J. Chapin, Dimitrios Katramatos, John Karpovich, and Andrew Grimshaw. Resource management in Legion. *Future Generation Computer Systems*, 15(5–6):583–594, 1999.

[CS93] Maria Calzarossa and Giuseppe Serazzi. Workload Characterization: A Survey. *Proceedings of IEEE*, 81(8):1136–1150, 1993.

[Fei02] D. G. Feitelson. Workload modeling for performance evaluation. *Lecture Notes in Computer Science*, 2459:114–141, 2002.

[Fis92] Paul A. Fishwick. SimPack: Getting Started with Simulation Programming in C and C++. In *Winter Simulation Conference*, pages 154–162, 1992.

[FK97] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[FK98] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.

[FKNT02] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The Physiology of the Grid - An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002.

[FKTT98] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A Security Architecture for Computational Grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[Fos01] Ian Foster. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[Fos02] Ian Foster. What is the Grid? A Three Point Checklist. Technical report, Argonne National Laboratory and University of Chicago, 2002.

[FTL+01] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: a computation management agent for multi-institutional grids. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium*, pages 55 – 63, San Francisco, CA, USA, August 2001. IEEE Computer Society Press.

[GFKH99] Andrew Grimshaw, Adam Ferrari, Fritz Knabe, and Marty Humphrey. Legion: an Operating Systems for Wide-area Computing. Technical Report CS-99-12, University of Virginia, March 1999.

[GN02] Aniruddha Gokhale and Balachandran Natarajan. GriT: A CORBA-based GRID Middleware Architecture. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*. Institute of Software Integrated Systems, IEEE, 2002.

[Gro00a] Object Management Group. CORBA Component Model. OMG Document ptc/2000-03-04, March 2000.

[Gro00b] Object Management Group. Fault Tolerant CORBA Specification. OMG Document ptc/2000-04-04, April 2000.

[GW+97] Andrew S. Grimshaw, Wm. A. Wulf, et al. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997.

[HE95] Kipp E. B. Hickman and Taher ElGamal. The SSL Protocol. RFC draft, Netscape Communications Corp., June 1995. Version 3.0, expires 12/95.

[HV99] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison Wesley Professional, 1999.

[IBM93] IBM Corporation, Kingston, NY. *IBM Load Leveler: User's Guide*, Septempber 1993.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[JR94] Jeffrey A. Joines and Stephen D. Roberts. Design of object-oriented simulations in C++. In *Winter Simulation Conference*, pages 157–165, 1994.

[KCM+00] Fabio Kon, Roy H. Campbell, M. Dennis Mickunas, Klara Nahrstedt, and Francisco J. Ballesteros. 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'9)*, pages 201–208, Pittsburgh, August 2000.

[KMY+05] Fabio Kon, Jeferson Robero Marques, Tomonori Yamane, Roy H. Campbell, and M. Dennis Mickunas. Design, implementation and performance of an automatic configuration service for distributed component systems. In *Software: Practice and Experience*, January 2005.

[Kon00] Fabio Kon. *Automatic Configuration of Component-Based Distributed Systems*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2000.

[KRL+00] Fabio Kon, Manuel Román, Ping Liu, Jina Mao, Tomonori Yamane, Luiz Claudio Magalhães, and Roy H. Campbell. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, number 1795 in LNCS, pages 121–143, New York, April 2000. Springer-Verlag.

[LLM88] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor : A hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, pages 104–111. IEEE Computer Society Press, June 1988.

[MK02] Jeferson Roberto Marques and Fabio Kon. Gerenciamento de Recursos Distribuídos em Sistemas de Grande Escala. In *Anais do XX Simpósio Brasileiro de Redes de Computadores*, pages 800–813, Maio 2002.

[NhCN99] Klara Nahrstedt, Hao hua Chu, and Srinivas Narayan. QoS-aware resource management for distributed multimedia applications. Technical Report 3-4, 1999.

[NHG01] Anand Natrajan, Marty A. Humphrey, and Andrew S. Grimshaw. Grids: Harnessing Geographically-Separated Resources in a Multi-Organizational Context. High Performance Computing Systems, June 2001.

[PACR02] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of HotNets–I*, Princeton, New Jersey, October 2002.

[Pow92] David Powell. Failure Mode Assumptions and Assumption Coverage. In Dhiraj K. Pradhan, editor, *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing (FTCS '92)*, pages 386–395, Boston, MA, 1992. IEEE Computer Society Press.

[Sch] D. Schmidt. ACE: The adaptive communication environment. Home page: `http://www.cs.wustl.edu/schmidt/ACE-papers.html`.

[SGH+97] D. Schmidt, A. Gokhale, T. Harrison, D. Levine, and C. Cleeland. TAO: a High-performance Endsystem Architecture for Real-time CORBA. *IEEE Communications Magazine*, 14(2), 1997.

[SSRB00] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture*, volume Vol. 2: Patterns for Concurrent and Networked Objects. John-Wiley & Sons, 2000.

[Teaa] Ethereal Development Team. *Ethereal: Network Protocol Analyzer*. Home page: `http://www.ethereal.com/`.

[Teab] PostgreSQL Development Team. *PostgreSQL: An object-relational database management system*. Home page: `http://www.postgresql.org/`.

[Yam00] Tomonori Yamane. The Design and Implementation of the 2K Resource Management Service. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, February 2000.

[YHK95] W. Yeong, T. Howes, and S. Kille. Lightweight Directory Access Protocol, March 1995. IETF RFC 1777.

[Zho92] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proceedings of the Workshop on Cluster Computing*, 1992.

Apêndice A

Equações

Média de uma amostra ($\bar{x}$)

$$\bar{x} = \frac{1}{n}\sum_{i=0}^{n} x_i \tag{A.1}$$

Onde n é o tamanho da amostra

Desvio padrão de uma amostra (s)

$$s = \sqrt{\frac{1}{n-1}\sum_{i=0}^{n}(x_i - \bar{x})^2} \tag{A.2}$$

Onde n é o tamanho da amostra

Intervalo de confiança (IC)

De acordo com o teorema do limite central, o intervalo de confiança para a média da população consiste em:

$$IC = \left(\frac{\bar{x} - z_{\frac{1-\alpha}{2}}}{\sqrt{n}}, \frac{\bar{x} + z_{\frac{1-\alpha}{2}}}{\sqrt{n}}\right) \tag{A.3}$$

Onde n é o tamanho da amostra;
α é o grau de confiança e
z é o valor x na abscissa de uma curva normal $N(0,1)$ tal que $P(X \geq x) = \frac{1-\alpha}{2}$.

Apêndice B

Consumo artificial de processamento

O programa construído para gerar sobrecarga artificial recebe o valor de uma média, a qual define o intervalo em que o gerador de carga consome o recurso de processamento da máquina e o intervalo em que deve permanecer inativo. Esse intervalo é resultado de uma distribuição exponencial.

Código-fonte do programa "workload-generator-cpu.c":

```
/* returns a random value between min and max */
float my_rand(float min, float max) {
  float r = min + max * rand() / (RAND_MAX + 1.0);
  return r;
}

float inverse_exponential(float mean) {
  float p;

  p = (float) log(my_rand(0.0, 1.0)) * (-mean);
  if (p < 1.0) {
    p = 1;
  }
  return p;
}

float next_arrival(float mean) {
  return inverse_exponential(mean);
}

float service_duration(float mean) {
  return inverse_exponential(mean);
}

int main(int argc, char *argv[]) {
  pid_t pid;
  int run_interval;
  int idle_interval;
  float mean;
  struct timeval tv;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s mean\n", argv[0]);
    exit(1);
  }
```

```
    /* seed for rand() */
    gettimeofday(&tv, NULL);
    srand(tv.tv_usec);

    mean = (float) atoi(argv[1]);

    for (;;) {
      /* waiting for a while */
      idle_interval = (int) rintf(next_arrival(mean));
      printf("Next process will run in %d seconds.\n", idle_interval);
      sleep(idle_interval);

      pid = fork();
      if (pid == -1) {
        fprintf(stderr, "fork failed: %s.\n", strerror(errno));
        exit(1);
      }

      if (pid == 0) {
        /* child consumes CPU */
        for (;;);
      }
      else {
        run_interval = (int) rintf(service_duration(mean));
        printf("process created with pid %d and executing for "
               "%d seconds.\n", pid, run_interval);
        sleep(run_interval);
        /* kill the child */
        if (kill(pid, SIGTERM) == -1) {
          fprintf(stderr, "terminating process %d failed: %s.\n",
                  pid, strerror(errno));
          exit(1);
        }
        printf("process %d terminated. ", pid);
      }
    }

    return 0;
  }
```

Apêndice C

Simulação de solicitação de execução de componentes

O programa construído para simulação da solicitação de instanciação e execução de componentes, ao ser acionado, tenta localizar um LRM na máquina local. Tendo sucesso nesta operação, ele efetua a requisição desejada ou realiza requisições subseqüentes, de acordo com os seguintes argumentos, fornecidos por linha de comando:

- `--component`: nome do componente a ser instanciado e executado. (`Sized1Component`, por exemplo);
- `--category`: categoria a que pertence o componente. A categoria define um conjunto de componentes definido por quem os provê. (`SizedComponent`, por exemplo);
- `--constraint`: restrições que devem ser satisfeitas para executar o componente. Deve estar de acordo com a *Resource Specification Language* (RSL), definida pelo CORBA. Por exemplo: `RAM_total > 512 and free_disk_space > 50`;
- `--loop`: se acionado, o programa executa indefinidamente, aguardando a cada requisição, um período calculado com base numa distribuição exponencial. Caso não seja acionado, o programa efetua uma só requisição e termina sua execução;
- `--mean`: média da distribuição exponencial utilizada no cálculo do valor de cada período de espera para realizar uma nova requisição.

Ao fazer uma requisição, o programa computa o intervalo de tempo que o sistema levou para fornecer a referência ao componente após sua instanciação ou para notificar que não foi possível encontrar uma máquina apropriada para executá-lo.

Apêndice D

Especificação das máquinas usadas nos experimentos

Nome da máquina	Processador	Memória volátil total (Mb)	Interface de rede (Mbps)	BogoMips
a202e02.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e04.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e05.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e07.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e08.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e09.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e10.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e11.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e13.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e16.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e18.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e19.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e20.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e21.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e24.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e26.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e27.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e28.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e29.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e31.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e32.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e33.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
a202e34.escet.urjc.es	Intel Pentium 4 2.6GHz	512	100	5177,34
grm1.homelinux.org	AMD Athlon XP 2600+	512	10	3792.89
grm2.homelinux.org	Intel Pentium III	256	100	1399,19
villa.ime.usp.br	AMD Athlon XP 1700+	512	100	2922,90
lobos.ime.usp.br	AMD Athlon XP 1700+	512	100	2922,90
motuca.ime.usp.br	Intel Pentium II	256	100	4122,38

Índice Remissivo

Resource Specification Language (RSL), 75
2K, 8, 12, 14, 21, 58, 59, 61, 62, 67
  apresentação, 12
  arquitetura e implementação, 14
  carga e execução de componentes
    intra-aglomerado, 18–19
  gerenciamento de recursos, 13–14
  inter-aglomerado, 21–35
  intra-aglomerado
    limitações, 21
    localização de componentes, 18–19
  serviços CORBA estendidos, 12
  serviços fundamentais para iniciação, 17–18

ACE, 14
aglomerado, 6–7
análise de desempenho, 37–57
  apresentação e interpretação dos dados, 50–57
    sistema de disseminação de informações (experimento 1), 50–52
    sistema de disseminação de informações (experimento 2), 52–54
    sistema de localização e execução de componentes (experimento 3), 54–57
  automatização do processo, 49
  carga aplicada, 48–49
    distribuição exponencial, 48–49
    no processamento, 43, 48, 52, 73–74
  máquinas usadas nos experimentos, 76
  método sistemático aplicado, 38–45
    carga aplicada, 41
    coleta das métricas, 45–47
    elaboração dos experimentos, 41–45
    fatores escolhidos, 40–41
    métricas usadas, 39
    parâmetros selecionados, 40
    serviços estudados, 38
    simulação para execução de componentes, 41, 44, 75
  sistema de coleta das métricas, 46–47
    funcionalidades, 46
  uma abordagem sistemática, 37–38

Condor, 60
Condor-G, 58, 60–61
CORBA
  trader, 14, 15
    limitação, 16

desempenho, 6
DSRT, 16, 18
dynamicTAO, 12, 14, 15

extensão do modelo intra-aglomerado, 21

Fault-Tolerant CORBA, 67
Foster, Ian, 8

Gerenciador de Recursos Global (GRM), 15
  intra-aglomerado, 15
    trader, 17
Gerenciador de Recursos Local (LRM), 15
  atualização da base de dados, 16
gerenciamento de recursos, 6
  com apoio a QoS, 8–10
  disseminação das informações
    intra-aglomerado, 16
  no 2K, 8, 13–14
  reserva de recursos, 14
  serviço de, 14
  tolerância a falhas, 16–17
gerencimento de recursos distribuídos, 7
Globus, 22, 58–60
grades computacionais, 7, 9, 21, 22, 58, 60–62
  definição, 8
  Globus, *veja* Globus
GriT, 62–63

inter-aglomerado
arquitetura, 22
disseminação de informações, 23–24
cenários em que ocorrem atualizações, 25
dica sobre estado dos recursos, 24
operações do GRM), 26
operações introduzidas, 26
processo de, 25–26
protocolo, 24–26
localização de recursos e execução de componentes, 28–34
escolha de candidatos, 29
iniciação dos serviços, 34
protocolo, 28–34
vantagem da configuração adotada, 23

Jain, Raj, 37

Legion, 58, 61–62

máquinas usadas nos experimentos, 76
middleware, 9, 61

Nimrod, 58

Open Grid Services Architecture (OGSA), 22
ORB reflexivo, *veja* dynamicTAO

p_1, 25
p_2, 25
p_3, 25
p_4, 25

qualidade de serviço (QoS), 7, 8
definição, 6

recursos distribuídos
exemplos de, 8
Repositório de Componentes, 15

TAO, 12, 14
trabalhos futuros
alternativa de comunicação inter-aglomerados, 64–65
diagnóstico da rede, 66
inclusão de recursos, 66
segurança, 65–66
tolerância a falhas, 66–67

Yamane, Tomonori, 14, 21