Gerenciamento de Recursos Distribuídos em Sistemas de Grande Escala

Jeferson Roberto Marques, Fabio Kon *
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
http://gsd.ime.usp.br
{jmarques,kon}@ime.usp.br

Palavras-chave: middleware CORBA, gerenciamento de recursos, qualidade de serviço, sistemas distribuídos escaláveis, computação em grade.

Abstract

With the popularization of computer networks in the past decade, local area networks composed of dozens of powerful machines became a commonplace in academic, commercial, and governmental institutions. These networks are often connected to the Internet and offer a wide variety of services to its users. Nevertheless, despite the almost two decades of research in distributed processing and load balancing, these systems are almost always used in a simplistic and inefficient way. A few services (such as file systems and the web) are distributed according to a conventional client/server model where the servers are almost always fixed. Most applications (such as text editors, calendars, spreadsheets, web browsers) are almost always executed locally in the client's machine even if it is not the most appropriate location to execute it and if there are other more powerful machines that are idle. A consequence of this behavior is that the resources in the distributed system are often underutilized and the quality of service offered to the user is not satisfactory.

The work presented in this paper revisits the concept of load balancing of user applications, presenting two new contributions. First, we address the problem with a novel architecture based on distributed objects and the CORBA standard. Second, we describe new protocols that are scalable to wide-area networks and can be used to integrate dozens or hundreds of heterogeneous local area networks, supporting the provision of quality of service. Thus, when logging into a machine connected to the network, the user gets access not only to the local machine but to the whole distributed system in all its power and diversity of resources.

*Esta pesquisa é financiada pela FAPESP, processo número 2001/03861-0, e CNPq, processo número 68.0118/01-2.

Resumo

Com a popularização das redes de computadores na última década, tornou-se lugar comum em instituições acadêmicas, comerciais e governamentais a existência de redes locais compostas por dezenas ou centenas de máquinas de médio poder computacional. Tais redes são muitas vezes conectadas à Internet e oferecem vários tipos de serviços aos seus usuários. No entanto, apesar das quase duas décadas de pesquisa na área de processamento distribuído e balanceamento de carga, estas redes são quase sempre utilizadas de uma forma simplista e ineficiente. Alguns poucos serviços (como, por exemplo, sistemas de arquivos e a *web*) são distribuídos através de um modelo cliente/servidor onde os servidores são quase sempre fixos. A maior parte das aplicações (como editores de texto, calendários, planilhas, navegadores da *web*) são quase sempre executadas localmente na máquina do cliente mesmo que esta máquina não seja a mais apropriada para isso e mesmo que haja uma enorme quantidade de recursos disponíveis na rede local para executar estas aplicações de forma mais eficiente. A conseqüência disto é que os recursos do sistema distribuído ficam muitas vezes ociosos ao mesmo tempo em que os usuários não obtêm uma qualidade de serviço satisfatória.

No trabalho aqui apresentado, buscamos resgatar o conceito do balanceamento da carga de aplicações dos usuários apresentando duas novas contribuições. Primeiramente, a arquitetura apresentada utiliza-se de um modelo de objetos distribuídos e baseia-se em um padrão da indústria (CORBA). Em segundo lugar, apresentamos novos protocolos que são escaláveis para redes de grande área, podendo ser usados para unir os recursos computacionais de dezenas ou centenas de redes locais heterogêneas oferecendo suporte para qualidade de serviço. Desta forma, ao utilizar uma máquina conectada à rede, o usuário tem ao seu dispor não só os recursos da máquina local mas também os recursos de todo o sistema distribuído em toda sua diversidade e poder computacional.

1 Introdução

Na década de 90, observou-se que, ao contrário do que ocorria no passado, o custo de uma sofisticada máquina paralela com alto poder computacional tornou-se muito mais alto do que o custo de uma rede local de computadores de pequeno porte com poder computacional equivalente. Este fato levou a novos modelos de computação chamados de Computação em Agrupamentos (*Cluster Computing*) ou, mais recentemente, Computação em Grade (*Grid Computing*) [FK99].

Infra-estruturas de middleware desenvolvidas nos últimos cinco anos como, por exemplo, Globus [FK98, SWMY00] e Legion [GW+97, GFKH99] permitem que uma coleção de máquinas heterogêneas distribuídas em agrupamentos fisicamente distantes, mas interconectadas por redes de longa distância como a Internet, trabalhem em conjunto para a resolução de problemas computacionalmente pesados. Os serviços oferecidos por este middleware permitem a localização das máquinas disponíveis em um determinado instante e auxiliam na execução de programas do usuário de forma altamente paralela em grandes grades contendo de algumas dezenas até vários milhares de máquinas.

Paralelamente, a pesquisa em sistemas operacionais distribuídos sempre buscou maneiras de combinar uma coleção de recursos computacionais distribuídos de uma forma homogênea e transparente para os usuários. Sistemas como V [Che88], Sprite [DO91] e Amoeba [TvRvS+90] permitem que usuários executem as suas aplicações em sistemas distribuídos da mesma forma em que executariam em uma máquina isolada. A distribuição dos diversos processos dos usuários para as várias máquinas de uma rede local é feita transparentemente pelo sistema sem que o usuário tenha que se preocupar com isso. Esta abordagem traz inúmeras vantagens e permite que os recursos da rede local sejam muito melhor aproveitados e que a carga seja balanceada entre as diversas máquinas. Ao contrário destes sistemas, em redes locais que utilizam sistemas operacionais tradicionais como UNIX ou Windows, mesmo que a grande maioria das máquinas estejam ociosas em um determinado instante, todas as aplicações iniciadas por um usuário são executadas em uma mesma máquina (a não ser que o usuário use comandos explícitos indicando onde cada aplicação deve ser executada).

Neste artigo, descrevemos um novo serviço de middleware baseado em objetos distribuídos que dá suporte para gerenciamento dinâmico de recursos em sistemas distribuídos. Este serviço engloba benefícios encontrados em 1) sistemas de computação em grade como Legion, 2) sistemas operacionais distribuídos como Amoeba e 3) sistemas de middleware baseados em objetos distribuídos com suporte para qualidade de serviço (QoS). Além disso, como o serviço é baseado em CORBA, ele pode interagir com outros serviços e aplicações CORBA e ser executado em cima de diversos sistemas operacionais e plataformas de hardware, incluindo PC/Linux, PC/Windows e Sparc/Solaris.

Em [KYH+01], descrevemos brevemente o primeiro protótipo deste serviço que, naquele momento, era limitado a um único agrupamento (*cluster*) em uma rede local. Agora, apresentamos uma nova versão que inclui um novo protocolo que torna o serviço altamente escalável permitindo que ele controle os recursos presentes em uma hierarquia de agrupamentos conectados por uma rede de grande área como a Internet. Apresentamos também, pela primeira vez, uma descrição detalhada dos protocolos envolvidos e os resultados obtidos em nossos primeiros experimentos com o sistema.

2 Arquitetura

O objetivo do Serviço de Gerenciamento de Recursos Distribuídos é oferecer a possibilidade de execução remota de componentes de software em uma coleção de computadores conectados através de redes locais e de redes de grande área. A escolha de onde as componentes são executadas é feita de forma transparente ao usuário e leva em conta características estáticas e dinâmicas do hardware e do software disponível no sistema distribuído. Além disso, usuários e programadores podem também especificar os requisitos de utilização de recursos das componentes de software. A partir desta especificação, o sistema se encarrega de localizar uma máquina apropriada para executar a componente e de garantir a qualidade do serviço (QoS) através da reserva dos recursos necessários.

Nesta seção, descrevemos a arquitetura do serviço em linhas gerais, mantendo uma abertura que permite a sua implementação em diferentes contextos usando tecnologias diversas. Na seção seguinte, no entanto, descrevemos uma implementação específica desta arquitetura que foi realizada através da tecnologia CORBA.

2.1 Arquitetura Intra-Agrupamento

Inicialmente, vamos analisar os elementos arquiteturais e os protocolos utilizados dentro de um único agrupamento de máquinas. Um agrupamento geralmente reside em uma única rede local e contém de 2 a 100 máquinas. A Figura 1 mostra a organização de um agrupamento.

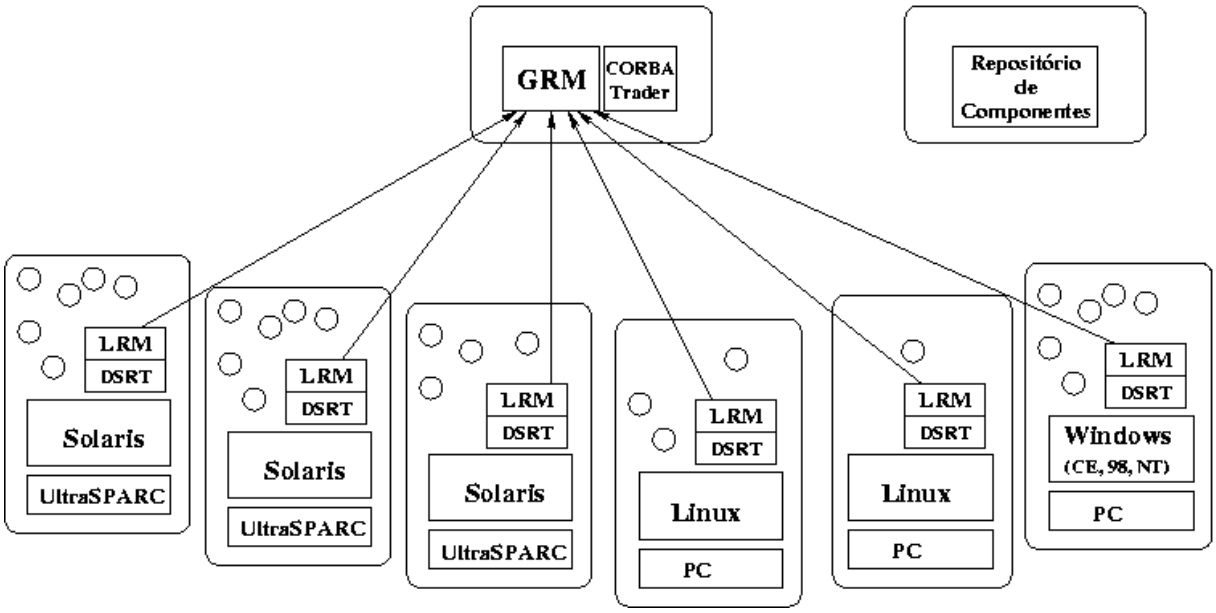

Figura 1: Organização de um agrupamento (*cluster*)

O serviço é formado por duas componentes principais: o Gerenciador de Recursos Local (LRM) e o Gerenciador de Recursos Global (GRM)[1]. Um serviço auxiliar importante é o Repositório de Componentes que armazena o código executável das componentes de software juntamente com informações sobre os requisitos para a execução daquela componente.

O LRM é responsável por gerenciar os recursos de uma única máquina. Para isso, ele interage com o sistema operacional (possivelmente através de uma biblioteca como veremos na Seção 3) e recebe solicitações para execução de novas componentes. Tais solicitações podem vir tanto de usuários locais que interagem diretamente com o LRM quanto de usuários ou serviços remotos cujas solicitações vem através do GRM.

O GRM, por sua vez, mantém uma visão global do agrupamento e é responsável pelo gerenciamento dos recursos do agrupamento como um todo. Para isso, o GRM mantém um banco de dados com informações sobre as máquinas do agrupamento e recebe atualizações periódicas sobre o estado destes recursos.

Nesta seção, apresentamos uma visão geral da arquitetura do serviço. Ele se baseia em dois protocolos fundamentais que descrevemos a seguir: o protocolo de atualização do banco de dados do GRM e o protocolo de reserva de recursos e execução de componentes. Em seguida, na Seção 3, apresentamos detalhes sobre a nossa implementação desta arquitetura que se utiliza do serviço *Trading* de CORBA e da biblioteca DSRT.

[1] As siglas LRM e GRM provêm do nome dos componentes em inglês, respectivamente *Local Resource Manager* e *Global Resource Manager*.

2.1.1 Execução de Componentes

O protocolo para a execução de novas componentes dentro de um agrupamento depende fortemente do estado dinâmico dos recursos do sistema. Como mostra a Figura 2, quando um usuário deseja executar uma nova componente de software[2], ele envia uma solicitação para o LRM local (flecha 1 da Figura 2); isto pode ser feito programaticamente, através de uma interface gráfica ou através de um interpretador de comandos interativo. Esta solicitação inclui o nome da componente a ser executada e, se desejável, informações sobre em qual tipo de máquina a componente deve ser executada e quais tipos de recursos a componente irá necessitar. Usuários leigos, provavelmente não saberiam (nem desejariam) fornecer todas estas informações; neste caso, o próprio programador da componente ou o administrador do sistema poderiam fornecer uma especificação dos requisitos da componente a ser armazenada no repositório de componentes.

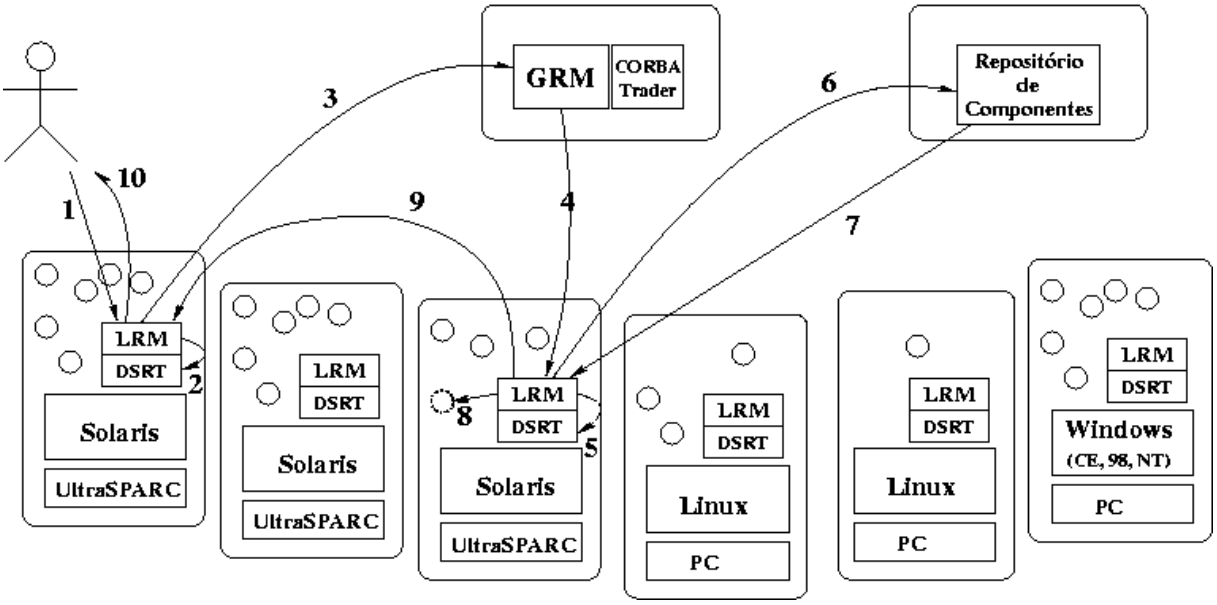

Figura 2: Protocolo de Reserva de recursos e execução de componentes

Quando o LRM recebe a solicitação do usuário local, ele primeiro examina os requisitos da componente e verifica se a máquina local tem a capacidade de atendê-los (2). Caso ela tenha esta capacidade, ele então verifica se a carga atual no processador local está em um nível satisfatório (por exemplo, menor do que 90% de utilização). Caso a carga esteja dentro do limite pré-estabelecido, o LRM então solicita ao repositório de componentes que lhe envie o código para aquela componente. Ao receber o código, este é executado em um novo processo na máquina local e, se necessário, recursos de hardware da máquina local (como parcelas da memória física e do processador) podem ser reservados para este processo a fim de garantir a qualidade do serviço para a nova componente.

Por outro lado, se o LRM decide que a execução da componente não deve ser local, ele repassa a solicitação para a execução da componente para o GRM (3). Este, por

[2]Neste artigo, a expressão "componente de software" é usada para se referir tanto a uma aplicação executada por um usuário (por exemplo, um editor de textos ou um navegador *web*) quanto a uma parte de uma aplicação ou sistema distribuído maior.

sua vez, consulta o seu banco de dados para descobrir qual, dentre todas as máquinas do agrupamento, seria o melhor candidato para executar tal componente. Para isso, o GRM leva em conta os requisitos de software e hardware da componente e o estado atual de utilização de recursos em cada máquina do agrupamento. Neste momento, o GRM repassa a solicitação para o candidato escolhido (4). O candidato verifica se ele realmente pode hospedar aquela componente localmente (5) e, caso possa, executa os mesmos passos descritos anteriormente para executar a componente, ou seja, baixa o código da componente do repositório (6,7) e a executa localmente (8). Após a inicialização da componente, o LRM envia uma mensagem de volta para o cliente que solicitou a execução da componente (9,10) contendo uma referência para a mesma de forma que o usuário possa interagir com ela.

Caso, no passo (5), o "melhor" candidato escolhido pelo GRM não possa hospedar a componente (por exemplo, porque os seus recursos de hardware já estão reservados para outros processos), o seu LRM envia um NACK para o GRM que, por sua vez, envia a solicitação para o segundo melhor candidato, e assim por diante. Como veremos em seguida, o GRM mantém uma visão aproximada da disponibilidade de recursos em todas as máquinas de seu agrupamento. Então, na grande maioria das vezes, o primeiro candidato escolhido por ele já é capaz de hospedar a componente.

2.1.2 Disseminação de Informações

Como vimos acima, é importante que o GRM mantenha uma informação atualizada sobre a utilização de recursos nas máquinas do agrupamento. No entanto, se projetássemos o sistema de forma que essas informações fossem atualizadas muito freqüentemente, isso levaria, com certeza, a uma grande sobrecarga e a um fraco desempenho do serviço, o que comprometeria a sua escalabilidade. Portanto, a estratégia que adotamos foi fazer com que o GRM não mantivesse informações precisas sobre o estado do agrupamento, mas sim, uma idéia *aproximada* do estado global do sistema. Desta forma, o banco de dados do GRM é utilizado apenas como uma dica (*hint*) sobre onde as componentes devem ser executadas, não buscamos um comportamento otimal do sistema, buscamos apenas um comportamento eficiente do sistema[3].

O protocolo de atualização das informações do banco de dados do GRM funciona da seguinte forma: periodicamente (por exemplo, uma vez por minuto), cada LRM consulta seu sistema operacional local para verificar a disponibilidade de recursos como memória física, memória virtual, largura de banda de rede e utilização do processador. Se houver uma mudança significativa na disponibilidade de algum destes recursos (por exemplo, uma mudança de mais de 10%) então, o LRM envia uma mensagem ao GRM informando o estado atual de todos os recursos. Se a mudança é considerada não muito significativa, nada é feito. Denominamos p_1 o intervalo entre duas consultas consecutivas ao sistema operacional.

Além disto, caso não haja nenhuma modificação significativa durante um prazo maior pré-determinado (por exemplo, a cada dez minutos), o LRM envia mesmo assim uma mensagem de atualização para o GRM. Este outro tipo de mensagem (chamada de *keep-alive*) é usada para que o GRM tenha uma idéia aproximada de quais máquinas estão disponíveis no sistema. Se uma máquina tiver problemas e se desligar da rede, este fato

[3]Mesmo porque se projetássemos um sistema que procurasse uma distribuição ótima das componentes, a sobrecarga seria tão grande que o desempenho se degradaria rapidamente.

é detectado pelo GRM graças à ausência do *keep-alive*. Denominamos P_2 este intervalo máximo entre o envio de mensagens ao GRM.

2.2 Arquitetura Inter-Agrupamentos

Os protocolos descritos acima funcionam bem para agrupamentos contendo até uma centena de máquinas. Como nosso objetivo é implementar um sistema capaz de atender potencialmente milhões de máquinas conectadas através de uma rede de grande área como a Internet, se faz necessário estender a arquitetura adotando-se uma hierarquia de agrupamentos. O objetivo maior de nosso design é oferecer o máximo de escalabilidade, mas sem perder o controle sobre os recursos do sistema.

A Figura 3 mostra a organização hierárquica de uma coleção de agrupamentos. Cada agrupamento pode tanto utilizar um repositório de componentes remoto quanto manter uma cópia local. A primeira opção facilita a administração dos repositórios enquanto que a segunda favorece a escalabilidade do sistema.

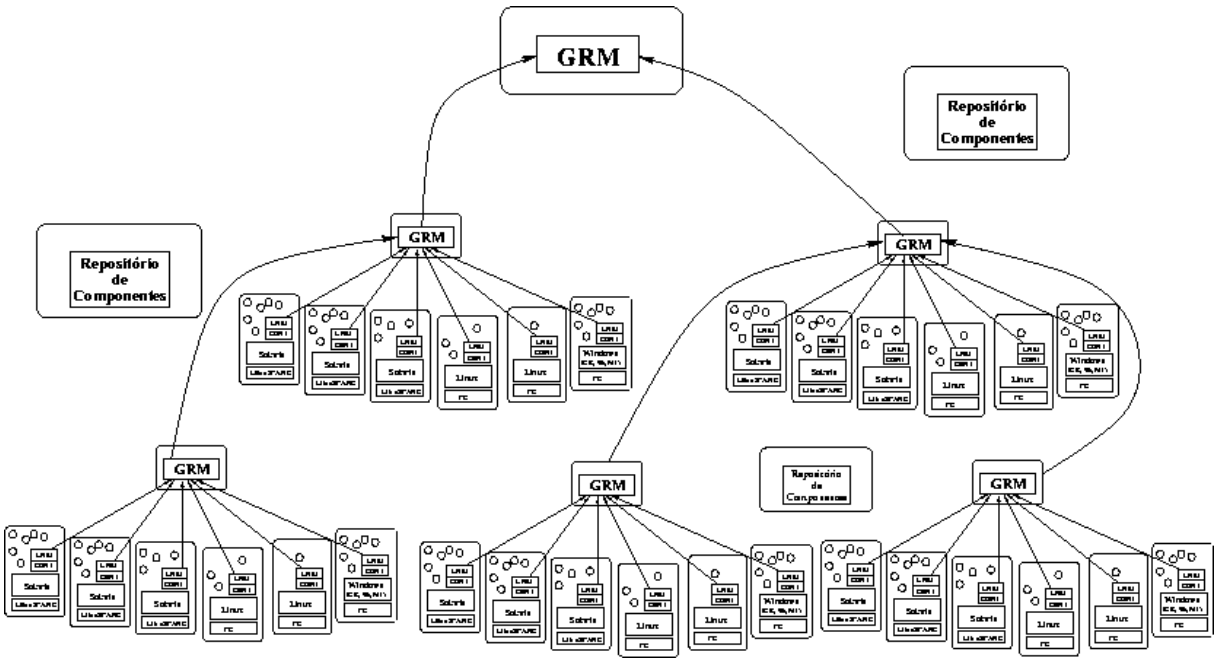

Figura 3: Hierarquia de agrupamentos

Os GRMs são organizados de forma hierárquica pelos administradores do sistema e o algoritmo de atualização é estendido de forma que GRMs em um nível da hierarquia enviem de tempos em tempos uma atualização do estado do seu agrupamento para o GRM do nível superior. No entanto, ao invés de enviar informações detalhadas sobre todas as máquinas do seu agrupamento (o que comprometeria a escalabilidade do sistema), cada GRM envia apenas uma consolidação (por exemplo, média aritmética e desvio padrão) da disponibilidade dos recursos nas máquinas do seu agrupamento. Assim, o GRM de um nó interior da árvore de GRMs mantém informações aproximadas sobre todos os agrupamentos que estão em sua sub-árvore. Um GRM envia uma atualização para o nível superior apenas quando há uma mudança significativa na consolidação dos recursos de

sua sub-árvore ou quando chega a hora de enviar um *keep alive* (por exemplo, a cada 10 minutos).

O algoritmo de reserva de recursos e execução de componentes também é estendido. Se um GRM percebe que não há nenhuma máquina em seu agrupamento capaz de executar uma certa componente, ele repassa a solicitação para um de seus vizinhos na hierarquia dos GRMs. Cada GRM intermediário possui informações sobre os recursos disponíveis nos seus vizinhos em níveis inferiores da árvore (chamados de "agrupamentos-filho"). Neste caso, o GRM usa estas informações para identificar o agrupamento filho que seria o melhor candidato para executar tal componente e executa um protocolo análogo ao protocolo de reserva de recursos e execução de componentes descrito em 2.1.1. Caso nenhum agrupamento filho seja capaz de executar a componente, o GRM repassa a solicitação para o seu GRM pai, de forma análoga ao protocolo intra-agrupamento onde o LRM repassa a solicitação para o seu GRM.

Espera-se que, numa situação normal, a grande maioria das componentes serão executadas no agrupamento local. Mas, caso o agrupamento esteja sobrecarregado ou caso os recursos de hardware exigidos por uma componente específica não existam no agrupamento local, o sistema se encarrega de buscar um local apropriado para executar a componente em outros agrupamentos.

3 Implementação

A fim de colocar em prática os conceitos apresentados acima, nosso grupo desenvolveu uma implementação do Serviço de Gerenciamento de Recursos Distribuídos baseado na tecnologia CORBA. Escolhemos esta tecnologia por quatro motivos principais: 1) CORBA é um padrão definido pela OMG (*Object Management Group*, um consórcio de dezenas de instituições públicas e privadas, acadêmicas e comerciais), 2) sua independência de plataforma de hardware e software e sua independência de linguagens de programação, 3) a possibilidade de reutilizar serviços padrão do CORBA implementados por outros grupos como, por exemplo, o serviço de nomes e o *Trader* e 4) a facilidade que outros serviços e aplicações CORBA teriam ao interagir com o nosso serviço, dado que bastaria utilizar um ORB CORBA qualquer e conhecer a interface do nosso serviço definida através da IDL (*Interface Definition Language*) padrão da OMG.

O LRM e a sua interface com o sistema operacional local é um dos pontos mais sensíveis no que diz respeito à portabilidade do sistema. A fim de evitar ao máximo as peculiaridades de cada sistema e ao mesmo tempo manter um controle fino sobre o gerenciamento de recursos, o LRM utiliza-se da biblioteca DSRT (*Dynamic Soft Real-Time*), desenvolvida na Universidade de Illinois em Urbana-Champaign [NhCN98], que dispõe de implementações para os sistemas Linux, Solaris e Windows. O DSRT oferece uma interface padrão para a obtenção de informações dinâmicas sobre o estado dos recursos em uma máquina, o que é utilizado pelo LRM no protocolo de atualização de informações. O DSRT provê também suporte para controle de admissão de novos processos baseado em seus requisitos, reserva de recursos (memória e processador) e escalonamento de tempo real flexível (*soft real-time*). Estas características são utilizadas pelo LRM no protocolo de reserva de recursos e execução de componentes de forma que quando o serviço executa uma componente em um determinado nó, ele oferece a garantia de que os recursos requisitados estarão disponíveis e que o escalonamento do processo contendo a componente obedecerá à especificação das necessidades da componente. A versão atual do DSRT ainda

não oferece suporte para reserva de largura de banda de rede e portanto a implementação corrente de nosso serviço ainda não leva em consideração os requisitos relativos à rede. Espera-se que as novas versões do DSRT ofereçam suporte para a rede; quando isto acontecer, deverá ser trivial estender o nosso sistema para utilizar este suporte uma vez que bastaria acrescentar mais um tipo de recurso a ser gerenciado.

O banco de dados do GRM é implementado como uma extensão do *Trader* padrão de CORBA. Para cada máquina de um agrupamento, o GRM armazena, em seu *Trader*, valores para atributos estáticos (plataforma de hardware, tipo de sistema operacional, velocidade do processador, quantidade total de memória física, virtual e espaço em disco) e dinâmicos (parcela utilizada do processador, memória física, memória virtual e disco). Além disso, o sistema se utiliza do Serviço de Nomes padrão de CORBA e do Repositório de Componentes desenvolvido para o sistema *2K* [KCM+00].

O código fonte e a documentação para o sistema são distribuídos livremente e podem ser obtidos em `http://gsd.ime.usp.br/software`.

4 Avaliação do Desempenho

A fim de analisar a sobrecarga causada por GRMs e LRMs em um sistema distribuído, efetuamos uma série de experimentos controlados em nosso laboratório. É nosso objetivo estabelecer a médio prazo um ambiente para testes compostos por dezenas de máquinas espalhadas em múltiplas redes locais localizadas em diferentes continentes. Como isso ainda não foi possível, apresentamos aqui o resultado de experimentos realizados com duas máquinas de nosso laboratório ligadas através de uma rede Ethernet de 100Mbps. Uma das máquinas (processador Pentium III 800MHz e 192MB de RAM) executa o servidor de nomes e o GRM enquanto que a outra máquina (com dois processadores Pentium III 750MHz e 768MB de RAM) executa várias instâncias do LRM para simular a carga gerada por vários clientes, cada um executando um LRM. Acreditamos que esta configuração apresenta uma boa aproximação do que aconteceria se tivéssemos várias máquinas em uma rede local uma vez que a carga na rede e no GRM seria praticamente a mesma.

4.1 Sobrecarga causada pelos LRMs nos clientes

Em uma situação normal, o LRM deveria ser configurado para consultar o estado dos recursos locais a cada 30 ou 60 segundos e para enviar atualizações ao GRM pelo menos uma vez a cada 1 ou 2 minutos (como vimos na Seção 2.1.2). No entanto, para avaliar a sobrecarga imposta às máquinas dos clientes, configuramos o LRM para realizar estas tarefas muito mais freqüentemente: consultas locais uma vez por segundo e envio de atualizações ao GRM pelo menos a cada 2 segundos (ou seja, $p_1 = 1$ e $p_2 = 2$). Mesmo assim, a utilização do processador pelo LRM ficou sempre abaixo de 0.1% sendo que a média em cinco experimentos foi de 0.04%, o que demonstra que a sobrecarga no uso do processador causada pelo LRM é desprezível.

Por outro lado, o nosso protótipo foi desenvolvido utilizando TAO [SC99] que é um ORB CORBA muito completo e poderoso e que, portanto, exige o uso de muita memória. Desta forma, o protótipo necessita de cerca de 12MB de memória nas máquinas clientes, o que em alguns casos pode ser uma sobrecarga significativa. Este problema poderia ser

solucionado com a utilização de ORBs desenvolvidos especificamente para máquinas com poucos recursos [RKC01] exigindo apenas alguns quilobytes de memória.

4.2 Sobrecarga causada pelo GRM no servidor

Em situações normais, o GRM deve ser executado em um servidor da rede local e atender entre 10 e 100 clientes. A Figura 4 apresenta o percentual de utilização do processador pelo GRM quando o número de LRMs é de 2, 8, 16, 32 e 64. Neste experimento, os LRMs foram configurados com $p_1 = 30$s e $p_2 = 60$s. Quando realizamos este experimento, tanto a rede quanto o processador estavam sendo utilizados exclusivamente para esta tarefa. Para simular variações significativas na utilização do processador e da memória dos clientes, executamos um programa escrito especificamente para causar perturbações no uso dos recursos a cada 50 segundos e, por conseqüinte, gerar mais mensagens do LRM para o GRM. Como demonstra a Figura 4, a carga imposta pelo GRM no processador do servidor é sempre muito pequena chegando a 3.6% quando há 64 LRMs. Isto demonstra que um GRM pode facilmente acomodar muitas dezenas de LRMs, não comprometendo a escalabilidade do sistema, e que a máquina que hospeda o GRM ainda pode ser utilizada para outras finalidades. Cada ponto no gráfico é a média de cinco repetições do experimento e as barras verticais representam o desvio padrão.

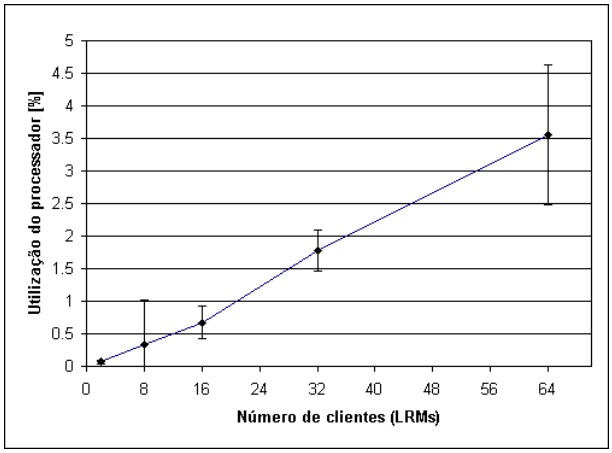

Figura 4: Sobrecarga gerada pelo GRM no servidor

O gráfico da Figura 5 mostra o percentual de utilização do processador pelo GRM. Mas, neste caso, fixamos o número de LRMs em 32 e 64 e variamos a dupla de parâmetros (p_1, p_2) dentro de (1,2), (5,10), (10,20), (30,60) e (60,120) segundos. Mesmo com 64 LRMs enviando mensagens de atualização ao GRM pelo menos a cada $P_2 = 2$ segundos (o que é muito acima do normal), a utilização do processador do servidor não passou de 36%. Quando o período de verificação dos recursos é de $P_1 = 60$ segundos e o número de clientes é 64 (o que seria um caso típico), o uso do processador fica em torno de 2.3% o que aponta para uma excelente escalabilidade do sistema.

4.3 Sobrecarga na rede

Para medir a sobrecarga na rede imposta pelas mensagens de atualização dos LRMs para o GRM, realizamos experimentos com 2 a 64 LRMs e fixamos $p_1 = 30$ e $p_2 = 60$. A Figura 6

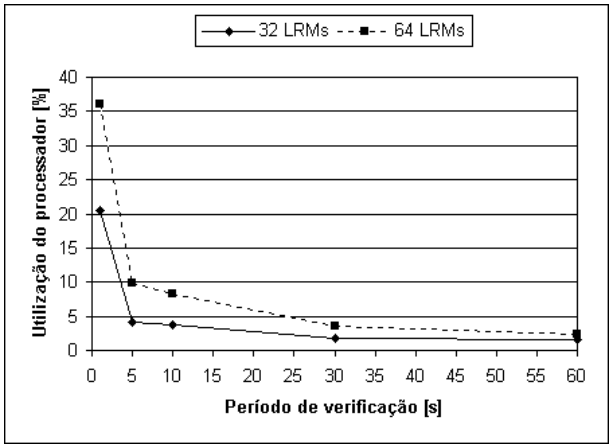

Figura 5: Sobrecarga gerada pelo GRM com 32 e 64 LRMs

apresenta a sobrecarga na rede em mensagens por segundo em dois tipos de experimentos: (1) com o sistema ocioso, sem mudanças significativas no uso dos recursos locais e (2) com perturbações geradas pelo programa desenvolvido para este fim. Novamente, podemos perceber que a sobrecarga imposta pelo sistema é pequena também na utilização da rede.

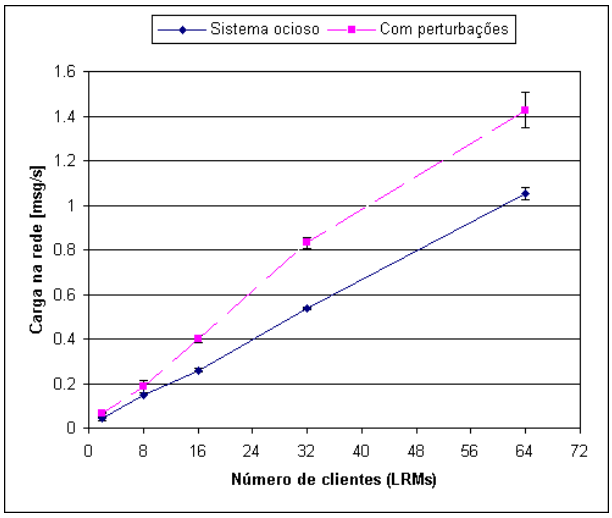

Figura 6: Tráfego gerado na rede ($p_1 = 30$, $p_2 = 60$)

4.4 Tempo para execução de componentes

A Figura 7 mostra o tempo para carga e execução de componentes de diferentes tamanhos. O tempo inclui todos os 10 passos descritos na Seção 2.1.1 e apresentados na Figura 2. Podemos notar que o tempo total gasto com uma componente de 49KB (354ms) foi apenas ligeiramente superior ao tempo gasto com uma componente de 10KB (298ms). Isto indica que a maior parte do tempo é gasta com as partes do protocolo que independem do tamanho da componente como, por exemplo, a localização da melhor máquina para execução da componente e a verificação e alocação de recursos. Esta parte do código não foi totalmente otimizada e poderá ter seu desempenho melhorado no futuro caso haja necessidade.

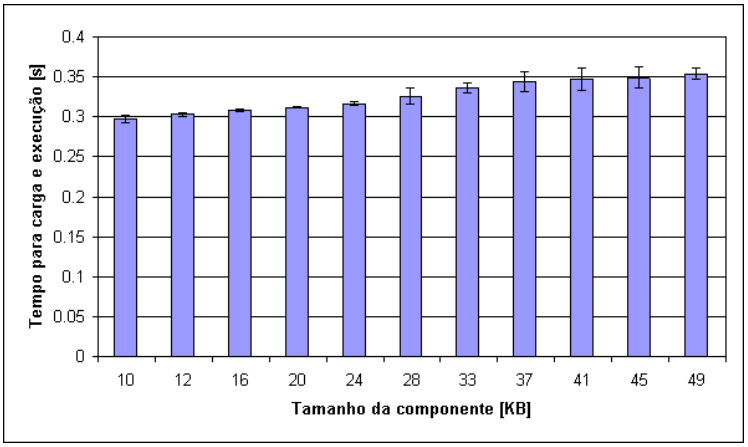

Figura 7: Tempo para execução de componentes

4.5 Experimentos Futuros

Os dados apresentados acima, mostram a sobrecarga causada pelo protótipo do nosso serviço para manter informações aproximadas sobre o estado global dos recursos da rede. Isto não leva em conta as atividades realizadas pelos usuários do sistema. No futuro próximo, realizaremos experimentos simulando a carga gerada por centenas de usuários solicitando a execução de várias componentes (aplicações) simultaneamente.

Finalmente, devemos também realizar experimentos para avaliar o desempenho dos protocolos inter-agrupamentos em redes de grande área.

5 Trabalhos Relacionados

A idéia da melhor utilização de recursos ociosos em redes locais através do balanceamento de carga surgiu em sistemas operacionais distribuídos como o V [Che88] e o Sprite [DO91]. Tais sistemas ainda apresentam uma vantagem em relação à nossa implementação atual que é a possibilidade de migração de processos de uma máquina para outra após o início de sua execução. No entanto, tais sistemas exigem uma rede homogênea (ao contrário do sistema apresentado neste artigo que é capaz de integrar diferentes arquiteturas de hardware e software) e não levam em conta os requisitos dos componentes, não oferecendo nenhum suporte para reserva de recursos e escalonamento de tempo real.

O sistema Globus [FK98, SWMY00] implementa uma grade computacional [FK99] integrando uma coleção heterogênea de recursos distribuídos em uma rede de grande escala. Ele oferece suporte para gerenciamento de recursos baseado em uma hierarquia de gerenciadores similar à proposta neste artigo. Mas, ao contrário de nosso sistema, Globus não oferece suporte para reserva de recursos, é baseado em um protocolo de comunicação proprietário e interfaces na linguagem C (ao invés de usar um middleware padrão como CORBA e interfaces independentes de linguagem definidas através de IDL).

Legion [GW+97, GFKH99] é o sistema para gerenciamento de recursos distribuídos mais similar ao nosso na medida em que também é baseado em uma arquitetura orientada a objetos e se utiliza de ORBs (*Object Request Brokers*) para comunicação. No entanto, Legion utiliza um middleware proprietário ao invés de CORBA e, assim como Globus, não oferece suporte para reserva de recursos.

6 Limitações e Trabalhos em Andamento

Atualmente estamos trabalhando para obter mais dados quanto ao desempenho do sistema, especialmente em relação à sua escalabilidade em redes de grande área. De posse desses dados, refinaremos a arquitetura e a implementação do serviço a fim de obter a melhor relação possível entre escalabilidade e controle fino sobre o gerenciamento.

Estamos particularmente interessados em estudar melhores alternativas para a consolidação das informações sobre os recursos disponíveis e em melhorias no algoritmo de disseminação de informações na hierarquia de agrupamentos.

Na implementação atual, os GRMs em nós interiores da hierarquia de agrupamentos não são capazes de receber informações diretamente de LRMs locais sendo necessário adicionar um GRM adicional no mesmo agrupamento para receber as informações dos LRMs locais. Este é um pequeno problema na implementação que será sanado rapidamente.

Finalmente, apesar da arquitetura contemplar o suporte para replicação dos GRMs em cada nível da hierarquia a fim de evitar pontos únicos de falha, ainda não exploramos devidamente este aspecto em nossa implementação de modo a oferecer um suporte adequado para tolerância a falhas. Detalhes sobre estes mecanismos não foram incluídos neste artigo por falta de espaço e serão o objeto de artigo futuro.

7 Conclusão

Este artigo estende o nosso trabalho anterior em gerenciamento de recursos distribuídos apresentando uma descrição detalhada de uma arquitetura escalável para gerenciamento de recursos distribuídos em redes de grande área. A arquitetura permite o balanceamento de carga de forma transparente aos usuários e possibilita a integração de vários agrupamentos locais heterogêneos em uma grande grade mundial. Mostramos uma implementação prática da arquitetura baseada em middleware padrão de objetos distribuídos (CORBA) oferecendo suporte para reserva de recursos e escalonamento de tempo real flexível. Os resultados experimentais demonstram que o serviço implementado, se configurado propriamente, impõe uma sobrecarga pequena às máquinas do sistema.

Agradecimentos

Agradecemos à equipe do projeto *2K* que ajudou na elaboração da versão inicial dos protocolos descritos neste artigo e a Tomonori Yamane que foi responsável pela implementação da primeira versão do serviço para um único agrupamento.

Referências

[Che88] David Cheriton. The V Distributed System. *Communications of the ACM*, pages 314–334, 1988.

[DO91] Douglis and Ousterhout. Transparent process migration: Design alternatives and the Sprite implementation. *Software - Practice and Experience*, 21(8):757–786, August 1991.

[FK98] I. Foster and C. Kesselman. The Globus Project: A Status Report. In *Proceedings of the IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[FK99] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, 1999.

[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.

[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.

[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.

[KYH+01] Fabio Kon, Tomonori Yamane, Christopher Hess, Roy Campbell, and M. Dennis Mickunas. Dynamic Resource Management and Automatic Configuration of Distributed Component Systems. In *Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'2001)*, pages 15–30, San Antonio, Texas, February 2001.

[NhCN98] Klara Nahrstedt, Hao hua Chu, and Srinivas Narayan. QoS-aware Resource Management for Distributed Multimedia Applications. *Journal of High-Speed Networking, Special Issue on Multimedia Networking*, 7:227–255, 1998.

[RKC01] Manuel Román, Fabio Kon, and Roy Campbell. Reflective Middleware: From Your Desk to Your Hand. *IEEE Distributed Systems Online*, 2(5), July 2001. Avaliable at `http://computer.org/dsonline/0105/features/rom0105_1.htm`.

[SC99] Douglas C. Schmidt and Chris Cleeland. Applying Patterns to Develop Extensible ORB Middleware. *IEEE Communications Magazine Special Issue on Design Patterns*, 37(4):54–63, May 1999.

[SWMY00] W. Smith, A. Waheed, D. Meyers, and J. Yan. An Evaluation of Alternative Designs for a Grid Information Service. In *Proceedings of the 9th IEEE Symposium on High-Performance Distributed Computing*, pages 185–192, Pittsburgh, August 2000.

[TvRvS+90] Andrew S. Tanenbaun, Robbert van Renesse, Hans van Staveren, Gregory J. Sharp, Sape J. Mullendar, Jack Jansen, and Guido van Rossum. Experiences with the Amoeba Distributed Operating System. *Communications of the ACM*, 33(12):47–63, December 1990.