# Uma Plataforma Distribuída com Balanceamento de Cargas para Servidores Web Baseada na Diferenciação de Serviços

**Antônio Serra[*1], Dominique Gaïti[2], Giovanni Barroso[3], Ronaldo Ramos[4], Jérôme Boudy[1]**

[1]Institut National des Télécommunications
9 rue Charles Fourier – 91011 - Evry Cedex, France

[2]LM2S – Université de Technologie de Troyes
12 rue Marie Curie – 10.010 - Troyes Cedex, France

[3]Universidade Federal do Ceará
Av. da Universidade, 2853 – Benfica - 60020-181 - Fortaleza - CE-Brasil

[4]Centro Federal de Educação Tecnológica - CE
Av. 13 de Maio 2081 Fátima - 60040-531 Fortaleza Ce Brasil

{antonio.de_barros_serra@int-evry.fr; dominique.gaiti@utt.fr; gcb@fisica.ufc.br; ronaldo@cefet-ce.br; jerome.boudy@int-evry.fr }

***Abstract.*** *Distributed services are now broadly available in the Internet, by means of applications such as e-learning platforms, telemedicine platforms, etc. To well assist a large number of users at the same time, these offered services must be distributed on a certain number of Web Servers Nodes. One possible solution to improve performance in these distributed applications is to make an efficient distribution of service avoiding concentration of several tasks on only one server. This paper presents a load-balancing platform that works based on the differentiation of services and increases application-level QoS (Quality of Service) in Web Server Clusters. We also present experimental results showing that the platform balances the imposed load and differentiates the QoS offered to different services.*

***Resumo.*** *Serviços distribuídos estão amplamente disponíveis na Internet, através de aplicações como plataformas de ensino a distância, plataformas de telemedicina, etc. Para atender bem a um número considerável de usuários ao mesmo tempo, estes serviços devem ser distribuídos em um certo número de Servidores Web. Uma possível solução para melhorar o desempenho destas aplicações é fazer uma distribuição eficiente dos serviços de forma a evitar a concentração de várias tarefas em um único servidor. Este artigo apresenta uma plataforma de balanceamento de cargas que trabalha baseada na diferenciação de serviços com o objetivo de fornecer níveis diferenciados de QoS (Qualidade de Serviço) em Clusters de Servidores Web. Também são apresentados resultados experimentais que mostram que a plataforma equilibra a carga imposta e que oferece diferentes níveis de QoS aos serviços disponibilizados.*

[*] Antonio Serra é financiado pela CAPES-Brasil.

## 1. Introdução

O número de usuários de serviços oferecidos pela Internet vem aumentando continuamente. Várias aplicações (plataformas de telemedicina, plataformas de educação à distância, plataformas de e-commerce, etc.) estão sendo disponibilizadas em servidores na WWW (World Wide Web). Em geral, estes ambientes são compostos de aplicações com diferentes características e precisam oferecer uma QoS (qualidade de serviço) mínima e diferenciada.

Ao nível de rede, existem várias soluções para a provisão de QoS no mundo IP: Podemos mencionar: DiffServ (Differentiated Services), IntServ (Integrated Services), MPLS (Multi-protocol Label Switching), ISLL (Integrated Service over Specific Link Layers) e RSVP (Resource Reservation Protocol). Estes mecanismos permitem a garantia de alguns parâmetros de QoS (perdas, retardo, variação do retardo, etc.) em redes do tipo IP.

Ao nível de aplicação, um balanceamento de cargas dinâmico e eficiente dos serviços na infra-estrutura disponível é uma boa solução para melhorar a QoS oferecida ao cliente (levando em conta o tempo de resposta). O balanceamento de cargas pode ser usado como um modo efetivo para evitar a concentração de tarefas em um único servidor. Em sistemas distribuídos, o balanceamento de cargas melhora o desempenho reduzindo o tempo de resposta do cliente. A disponibilidade dos serviços pode ainda ser aumentada utilizando-se mecanismos de tolerância à falhas como a replicação.

Há um número considerável de trabalhos no domínio do “balanceamento de cargas" em servidores. Estes trabalhos são bastante heterogêneos. Algumas pesquisas se dedicam especificamente a busca de algoritmos para permitir a melhor distribuição possível dos recursos disponíveis [1]. Outras levam em consideração a especificação de serviços que podem integrar as funcionalidades de balanceamento de cargas ao middleware [2][3]. Existem ainda trabalhos que estudam o uso de agentes para a realização de um balanceamento de cargas eficiente [4]. Neste trabalho, é focalizada uma solução de balanceamento de cargas transparente introduzida no nível de aplicação.

Neste artigo é apresentada uma plataforma que realiza o balanceamento de cargas em Servidores Web e que se baseia na diferenciação de serviços para alocar de forma diferenciada os recursos disponíveis. A plataforma agrupa servidores em diferentes clusters Web de acordo com as classes de serviços estabelecidas. Cada cluster Web é responsável por atender à uma classe específica de serviços e por garantir uma QoS medida por meio do "coeficiente de reatividade” do cluster. As requisições dos clientes são atendidas de forma diferenciada, de acordo com a classe de serviço à qual elas pertencem. A plataforma permite a coexistência de objetos distribuídos e serviços Web e balanceia a carga imposta por ambos entre os servidores disponíveis. São apresentados tambem resultados de experimentos que mostram que a plataforma equilibra a carga imposta ao sistema e oferece diferentes níveis de QoS de acordo com a classe de cada serviço.

Este artigo está organizado da seguinte forma: Na seção 2, são apresentados alguns trabalhos relacionados. Na seção 3, são discutidas algumas características da solução de balanceamento de cargas proposta. Na seção 4, é apresentada uma especificação da plataforma. A implementação da plataforma é apresentada na seção 5. Na seção 6, são mostrados alguns resultados de experimentos relativos ao balanceamento de cargas e de

experimentos que mostram a capacidade da plataforma em oferecer níveis diferenciados de QoS para as requisições processadas. Finalmente, na seção 7, conclusões e perspectivas de trabalhos futuros são discutidas.

## 2. Estado da Arte

Existem trabalhos na área de balanceamento de cargas em sistemas distribuídos que merecem uma atenção especial. Alguns dão prioridade aos aspectos de distribuição [5] [2], outros tentam justificar a importância do uso de inteligência para resolver o problema proposto [6] [7] e mais recentemente alguns trabalhos mostram bons resultados com o uso da tecnologia agente [3] [4]. Ainda existem vários outros trabalhos neste domínio de pesquisa mas eles não estão estreitamente relacionados com a proposta aqui apresentada [8][9][10].

Em [5] é apresentado um sistema denominado “Realize” que simplifica o desenvolvimento de aplicações complexas separando a programação da aplicação do gerenciamento de recursos em aplicações CORBA de tempo real. O sistema replica objetos CORBA para prover uma alta disponibilidade e uma alta tolerância à falhas. Através do monitoramento dos recursos e do comportamento dos objetos das aplicações, “Realize” aloca objetos a processadores e migra objetos entre processadores para fazer o balanceamento das cargas.

Em [2] é apresentado o projeto de um serviço em nível de middleware que provê facilidades compartilhadas pelas aplicações para alcançar uma escalabilidade desejada provendo tolerância à falhas com adaptabilidade e propriedades de balanceamento de cargas. O projeto proposto não é restrito a uma tecnologia de distribuição especifica (CORBA, RPC, etc.), mas permite o uso transparente deste serviço por aplicações que foram escritas especificamente para estas tecnologias, minimizando a perda de desempenho causada à aplicação por frameworks específicos.

No campo das proposições que utilizam inteligência para prover o balanceamento de cargas, em [11] são apresentados o projeto e a implementação de um sistema de balanceamento de cargas dinâmico baseado em lógica fuzzy incorporado a um ambiente de computação baseado em objetos distribuídos. A solução proposta está baseada em um controlador de lógica fuzzy que informa ao objeto cliente qual o serviço mais apropriado de forma que a carga se mantenha equilibrada entre os servidores disponíveis. Eles escolheram a tecnologia "JINI" para construir o middleware experimental. O sistema de balanceamento de cargas é baseado em regras implementadas de forma concisa e fácil. O uso de um protótipo baseado em JINI foi considerado importante para a realização de estudos experimentais.

Em [7] uma unidade de classificação fuzzy é proposta como a base do projeto de um sistema de balanceamento de cargas inteligente para servidores de objetos distribuídos que utiliza o modelo de referência CORBA. O sistema proposto é baseado no padrão de serviço de nomes do CORBA e repousa em uma base de regras fuzzy. O sistema leva em conta os diferentes índices de desempenho dos servidores bem como a variação da qualidade das conexões de rede entre os mesmos.

Uma abordagem baseada em agentes moveis é apresentada em [4]. Uma técnica de alocação de informação autônoma é proposta. Nós e agentes móveis interagem localmente para estimar o nível de carga de trabalho dos nós e determinar o novo

volume de informação que deve ser alocado para satisfazer as necessidades atuais do usuário. Neste trabalho a efetividade em reduzir o tempo médio de resposta dos usuários foi provada através de simulações.

Na Universidade do Arizona um Sistema de Balanceamento de Cargas Dinâmico baseado em agentes tem sido desenvolvido para melhorar a QoS do usuário final. Em [3] é apresentado um framework para o balanceamento de cargas dinâmico baseado em agentes desenvolvido para clusters heterogêneos de computadores. O DASH (Dynamic Agent System for a Heterogeneous Cluster) é uma arquitetura intermediaria que se localiza sobre o nível do sistema operacional que provê dinamicamente para "n" tarefas um escalonamento não preemptivo de tarefas, controle de replicação e tolerância a falhas. O sistema configura dinamicamente os esquemas de balanceamento de cargas dependendo do estado corrente do cluster heterogêneo. Os serviços DASH são implementados através de agentes que são executados em cada servidor e que colaboram dinamicamente para estabelecer um conhecimento global dos recursos e estados do sistema. Baseado neste conhecimento dinâmico global, eles usam uma combinação de medidas de cargas e estimações estatísticas de medidas para escalonar os processos e então balancear as cargas através de todos os computadores do cluster. Um aumento significante de desempenho é alcançado utilizando-se a combinação de medidas de cargas com o algoritmo de predição estatística de carga. Eles concluiram que a construção em tempo real do mecanismo de escalonamento é mais interessante e pode ser melhorado se os requisitos de hardware do servidor forem modelados em seguida.

Alguns problemas ainda estão em aberto nas estratégias propostas. No que concerne a escalabilidade, muitos esquemas dependem fortemente do conhecimento global da distribuição das cargas e não são extensíveis para um grande número de servidores. Detalhes ligados ao projeto de um sistema de balanceamento de cargas genérico ainda não foram abordados com a profundidade necessária e muitos esquemas propostos são dirigidos a uma aplicação particular.

## 3. Características da Abordagem Proposta

Algumas soluções de balanceamento de cargas existentes requerem uma adaptação do "middleware", outras exigem a adaptação da aplicação desenvolvida. Isto gera alguns inconvenientes; O primeiro para o provedor de serviços que é obrigado a utilizar um middleware específico em sua plataforma; e o segundo para o desenvolvedor de aplicações que precisa adaptar a aplicação aos respectivos mecanismos de balanceamento de cargas. Na plataforma apresentada neste artigo foi utilizada uma estratégia que evita estes inconvenientes. A solução proposta é transparente ao provedor de serviços e ao desenvolvedor de aplicações. Os serviços de administração da plataforma geram os mecanismos necessários ao balanceamento de carga durante a instalação dos mesmos.

Em algumas aplicações é necessário associar níveis diferentes de QoS para os serviços oferecidos. Por exemplo, no contexto da telemedicina, em um sistema de monitoramento de pacientes existem tipos diferentes de usuários (pacientes, médicos, enfermeiras, etc.) e de serviços (processamento de sinais, solicitação de informações sobre o paciente, etc). Neste tipo de sistema todos os serviços são importantes, porém a prioridade de processamento de sinais vitais do paciente é maior que a prioridade do

atendimento a simples pedidos de informações cadastrais. A plataforma aqui apresentada permite o oferecimento de diferentes níveis de QoS para serviços específicos através de uma alocação diferenciada de recursos de processamento. Parâmetros de QoS são associados a classes diferentes de serviços. Quando um cliente faz uma requisição para um serviço, o sistema sabe qual a QoS que deve ser provida para esta requisição.

Novos paradigmas de software vêm mudando as metodologias de desenvolvimento de aplicações. As tecnologias orientadas a objetos são hoje extensamente utilizadas por desenvolvedores de aplicações. Tecnologias para desenvolvimento de sistemas distribuídos facilitam a comunicação entre elementos de software diferentes dentro da rede. Na Internet os serviços podem ser vistos como um conjunto de objetos distribuídos que trabalham conjuntamente para executar tarefas específicas. Nesta plataforma, serviços Web coexistem com objetos distribuídos e os mecanismos de balanceamento de carga levam em conta ambos os elementos.

Qualidade de Serviço é o efeito coletivo da ação de um serviço, o qual determina o grau de satisfação dos usuários do serviço [12]. No nível de rede, a QoS é medida através de parâmetros específicos tais como: perda de pacotes, retardo e variação do retardo. A nível de aplicação, a carga de um nó é diretamente relacionada com o tempo de resposta das aplicações que funcionam neste nó. Portanto, no nível de aplicação, a QoS percebida pelos usuários é diretamente afetada pelos níveis de carga dos servidores.

## 4. Especificação da Plataforma

A plataforma (Figura 1) é composta por um conjunto de elementos básicos: “Class Switch”, “Cluster Gateways” e “Web Server Nodes”. O “Class Switch” é responsável pela classificação e pelo controle de admissão de novas requisições de clientes. Ele recebe requisições HTTP, identifica a classe do serviço e envia cada requisição para um “Cluster Gateway” específico. O “Custer Gateway” escolhe o servidor Web menos sobrecarregado para processar a requisição enviada pelo “Class Switch”.

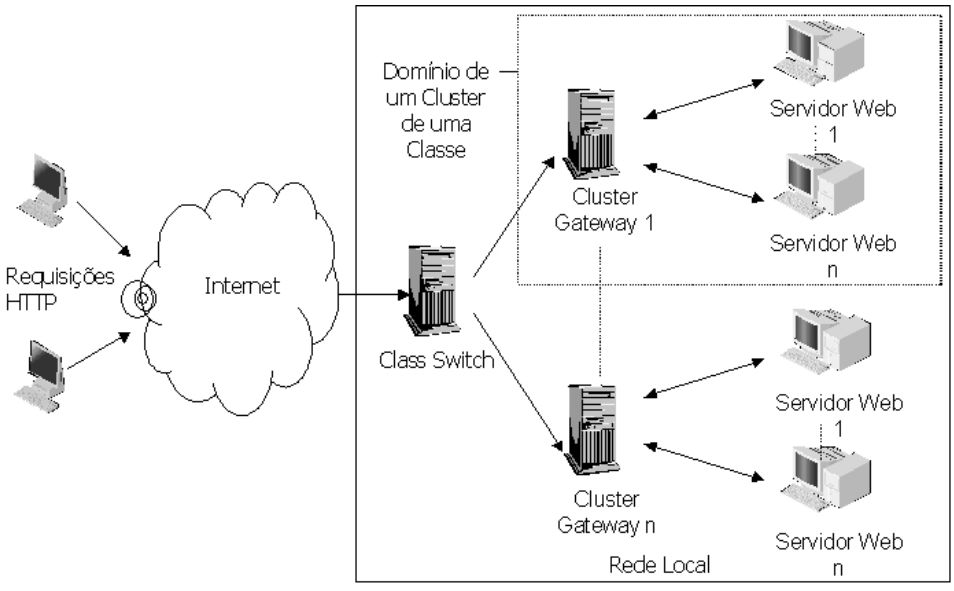



Figura 1 – Visão Geral da Plataforma

Os serviços são instalados em um certo número de servidores Web. Estes serviços podem ser compostos por serviços Web e objetos distribuídos. Cada Servidor Web possui um agente monitor que monitora a sua carga. O agente monitor usa um "Coeficiente de Reatividade" [13] que fornece uma idéia da tendência da carga do servidor. Esta técnica mede o tempo que uma tarefa espera para ter acesso ao tempo de CPU. O tempo de espera depende diretamente da carga do servidor. A seção 5.1 explicará com mais detalhes esta técnica.

No "Cluster Gateway", um componente denominado CRM (Cluster Resource Manager) periodicamente solicita informações da carga de cada servidor Web pertencente ao domínio do cluster. Ele estima a carga do cluster baseado nesta informação. Um componente denominado RG (Request Gateway) distribui cada requisição HTTP que chega para um servidor Web específico baseado nas informações de carga do CRM. Dentro do cluster de uma classe de serviços específica, as mensagens trocadas entre os objetos distribuídos são também redirecionadas por um componentes denominado MOG (Message Object Gateway), que também usa a mesma informação de carga fornecida pelo CRM para manter o balanceamento de cargas dentro do cluster.

A plataforma oferece diferentes níveis de QoS de acordo com cada classe de serviço. Requisições que chegam são classificadas em quatro classes diferentes de serviços: "gold", "silver", "bronze" e "best effort". O administrador da plataforma associa a cada classe de serviços uma carga máxima que pode ser atingida pelo cluster da classe. No "Class Switch" um componente denominado GRM (Global Resource Manager) solicita e mantém informações de carga de cada cluster. Quando uma requisição HTTP é recebida, o "Class Switch" identifica a classe da requisição e verifica a carga corrente do cluster da classe. Se o cluster pode garantir a QoS especificada para aquela classe de serviço o Class Switch redireciona a requisição para o cluster apropriado senão ele devolve uma mensagem recusando a requisição.

## 5 – Implementação da Plataforma

Um prototipo da plataforma proposta foi implementado utilizando-se a tecnologia Java (Figura 2). Os componentes da plataforma foram modelados como objetos distribuídos Java e serviços Web. O RMI (Remote Method Invocation) [14] Java foi utilizado para dar suporte à comunicação entre os objetos distribuídos e o Tomcat 5 Servlet/JSP [15] foi usado como container para os serviços Internet.

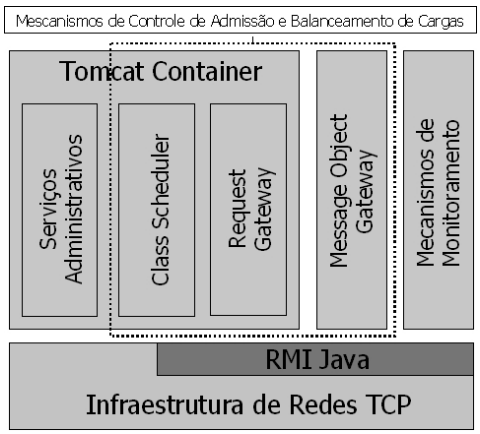



Figure 2 – Tecnologias de Implementação

Os elementos da plataforma são classificados como: Mecanismos de Monitoramento; Mecanismos de Controle de Admissão e Balanceamento de Cargas e Serviços Administrativos.

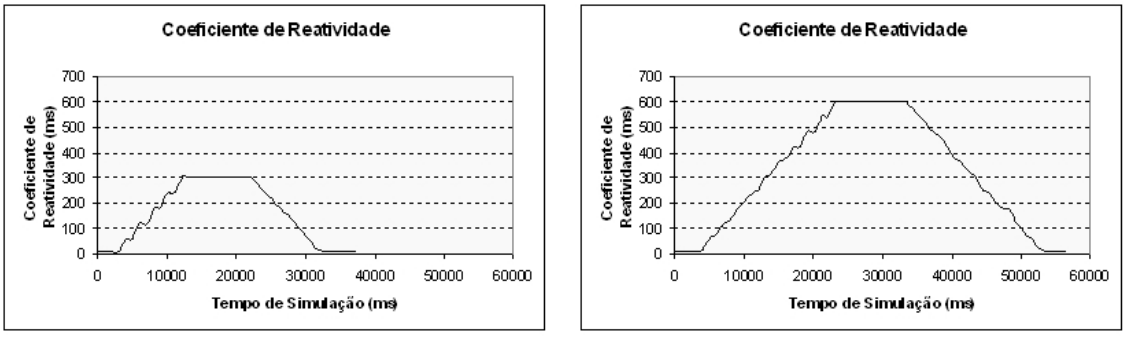
### 5.1 – Mecanismos de Monitoramento

Os mecanismos de monitoramento são compostos por três elementos básicos: o MA (Monitor Agent), o CRM (Cluster Resource Manager) e o GRM (Global Resouce Manager). O MA monitora e estima a carga de cada servidor Web e o CRM monitora e estima a carga de um cluster inteiro. O GRM solicita e mantém a informação de carga de cada CRM, ou seja, de cada cluster.

O MA possui três funções básicas. A primeira é de registrar o servidor Web em um domínio específico, ou seja, em um cluster de uma determinada classe de serviços. Ele envia uma mensagem solicitando o registro para o CRM responsável pelo domínio da classe pedindo para adicioná-lo ao domínio.

A segunda função do MA é de monitorar a carga do servidor Web. O MA usa uma técnica de monitoramento baseada em um "Coeficiente de Reatividade" [13]. Esta técnica dá uma idéia da tendência de carga do servidor estimando o tempo de espera de uma tarefa pela CPU. O MA possui um "thread" que passa o controle da CPU para um outro processo e espera ser novamente "acordado" pelo sistema. O tempo de espera do "thread" depende diretamente da carga do servidor. Para eliminar variações momentâneas da carga esta operação é repetida diversas vezes. Finalmente, é calculada a média do tempo de espera pela CPU (o "coeficiente de reatividade") que está diretamente relacionada com a carga do servidor.

Na Figura 3, pode-se observar o comportamento do "Coeficiente de Reatividade" de um servidor submetido a uma variação de carga. O servidor é um Pentium IV, 1.6GHz, 256 Mb de RAM utilizando o Microsoft Windows 2000 (Service Pack 4). No primeiro experimento (Figura 3.a), foi instalado no servidor um serviço que gasta 20000ms para ser processado em uma "condição ideal" (quando o servidor tem a carga mínima). No gráfico pode-se observar o comportamento do coeficiente de reatividade do servidor quando 5 requisições são enviadas pelo cliente em intervalos de 2000ms.




**(a)** **(b)**

Figura 3 – Comportamento do "Coeficiente de Reatividade"

No segundo experimento, foi instalado no servidor um serviço que gasta 30000ms para ser processado em uma condição ideal. Na Figura 3.b, encontra-se o comportamento do coeficiente de reatividade do servidor quando 10 requisições são

enviadas por clientes em intervalos de 2000ms. Pode-se observar que em ambos os casos o "Coeficiente de Reatividade" medido é proporcional à carga imposta ao sistema.

A terceira funcionalidade básica do MA é de responder ao CRM quando este solicita periodicamente informações sobre a carga de cada servidor Web.

Cada "Class Cluster Domain" executa um CRM (Cluster Resource Manager) que solicita informações de carga de cada servidor Web em intervalos constantes de tempo. Ele analisa todos os "coeficientes de reatividade" dos servidores do cluster e faz uma estimativa para a carga do cluster inteiro.

O GRM (Global Resource Manager) foi concebido por questões de escalabilidade. Os elementos "Class Scheduler" e "Cluster Gateway" poderiam ser instalados em um mesmo servidor na rede. Isto diminuiria a troca de mensagens entre os elementos da plataforma através da rede. No entanto, se o número de servidores do cluster for muito grande, o servidor que contém ambos os elementos pode ficar sobrecarregado. Para aumentar a escalabilidade do sistema foi considerado importante oferecer a oportunidade de manter o "Class Scheduler" e o "Cluster Resource Manager" em servidores diferentes dentro da plataforma (a implementação atual permite as duas possibilidades). No entanto, quando o "Class Scheduler" e o "Cluster Resource Manager" são colocados em servidores diferentes, aumenta-se o tráfego de mensagens na rede visto que uma comunicação entre os dois servidores é necessária sempre que o "Class Scheduler" precisa tomar uma decisão que depende do nível de carga dos servidores. Para minimizar esta comunicação o GRM solicita periodicamente as informações de carga de cada "Class Cluster" e mantém estas informações disponíveis localmente para o "Class Scheduler".

## 5.2 – Mecanismos de Controle de Admissão e Balanceamento de Carga

Os mecanismos de controle de admissão e balanceamento de cargas são compostos por três elementos: o CS (Class Scheduler), o HRG (HTTP Request Gateway) e o MOG (Message Objetc Gateway) (Figura 4). O CS é responsável pela classificação e pelo controle de admissão de requisições HTTP que chegam. O HRG envia requisições HTTP para o servidor menos carregado do cluster. O MOG é responsável pela interceptação e redirecionamento das mensagens trocadas entre os objetos distribuídos dentro de um cluster de forma a manter a carga balanceada entre os diversos servidores.

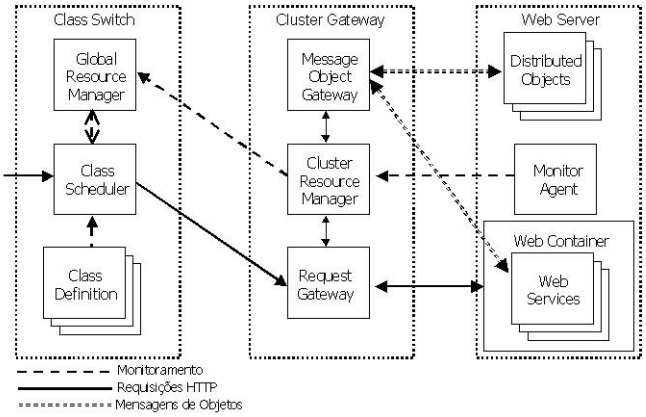



Figure 4 – Comunicação entre os elementos da plataforma

O CS recebe requisições que são enviadas pelos clientes. Depois de classificar a requisição identificando a classe de serviços a qual ela pertence, ele verifica junto ao GRM qual é a carga atual do cluster da classe e analisa se o cluster pode garantir o coeficiente de reatividade máximo definido para aquela classe. Se o cluster pode garantir a QoS especificada para a classe o CS redireciona a requisição para o respectivo "Cluster Gateway", senão ele retorna uma mensagem para o cliente informando que a QoS requerida para aquela classe de serviço não pode ser garantida. É importante salientar que arquitetura utiliza um mecanismo de balanceamento de cargas e admissão de controle que se baseia na realocação dinâmica de recursos [16], ou seja, enquanto existem recursos disponíveis em outras classes, estes recursos são realocados para classes em estado crítico. Isto evita, na medida do possivel, a rejeição de requisições. Esta realocação de recursos é feita respeitando a QoS que foi garantida para as classes que prioritariamente detêm os recursos.

O RG (Request Gateway) recebe as requisições redirecionadas pelo "Class Scheduler". Depois de receber a requisição ele decide qual servidor Web ira processá-la baseando-se na informação de carga mantida pelo CRM.

A plataforma permite a coexistência de serviços Web e de objetos distribuídos que podem interagir conjuntamente para prover serviços aos clientes. Quando um objeto distribuído é instalado na plataforma, o serviço de administração utiliza a interface de comunicação distribuída do objeto para automaticamente criar seu "Message Gateway". O MOG (Message Objetc Gateway) intercepta as invocações de métodos remotos dentro de um cluster específico e, baseado na carga informada pelo CRM, ele redireciona as invocações para o servidor menos carregado. Desta maneira, se um serviço Web utiliza objetos distribuídos para resolver suas subtarefas, a carga imposta por estas subtarefas também será balanceada entre os servidores do cluster.

### 5.3 – Serviços Administrativos

Um conjunto de serviços foi desenvolvido para permitir a administração da plataforma. Eles são agrupados em: Serviços de Instalação, Serviço de Definição de Classes e Serviços de Monitoramento dos Clusters.

Usando os serviços de instalação o administrador da plataforma pode instalar serviços Web e objetos distribuídos na plataforma. Os Serviços são replicados na plataforma e os mecanismos de balanceamento de carga são automaticamente criados.

O administrador pode definir classes de serviços, bem como as características de cada classe, utilizando o serviço de definição de classes. As características de cada classe estão diretamente relacionadas com o "coeficiente de reatividade" requerido para cada requisição de cliente pertencente àquela classe. O administrador do sistema associa um valor máximo para o coeficiente de reatividade de cada classe de serviço. A classe "gold" tem o melhor coeficiente de reactividade, a "silver" tem o segundo melhor e a "bronze" tem o pior coeficiente. A classe "best effort" não tem nenhuma garantia, ou seja não tem um coeficiente de reatividade máximo estabelecido.

Os serviços de monitoramento de carga exibem informações estatísticas (taxa de requisições recusadas, média do coeficiente de reatividade por cluster, etc.) sobre todo o sistema. Usando estas informações o administrador do sistema pode redefinir a estratégia de distribuição dos recursos entre os diversos clusters.

## 6 – Experimentos de Balanceamento de Carga

Para verificar a capacidade da plataforma em balancear a carga dentro de um cluster específico foram realizados alguns experimentos em um ambiente de teste real. Na Figura 5, é apresentada a arquitetura de hardware utilizada para a realização dos experimentos.

### 6.1 – Arquitetura de Hardware e Software

Um cenário heterogêneo foi utilizado (sistemas operacionais e configurações de hardware differentes). Os equipamentos utilizados para conduzir os experimentos foram:

- Servidor Web 1 e servidor Web 2, PCs genéricos contendo um processador Intel Pentium IV 1,6GHz, 264Mo de RAM, e uma placa de rede Ethernet de 100Mbs. O sistema operacional utilizado foi o Windows 2000 (Service Pack 4);
- Como "Cluster Gateway", um notebook com um processador Intel Celeron 1,1GHz, 264Mo de RAM, e uma placa de rede Ethernet de 100Mbs. O sistema operacional utilizado foi o Windows XP;
- Como equipamento cliente foi utilizado um PC genérico, com um processador Intel Pentium III 350 MHz, 64Mo of RAM, e uma placa de rede Ethernet de 100Mbs. O sistema operacional utilizado foi o Windows 98;

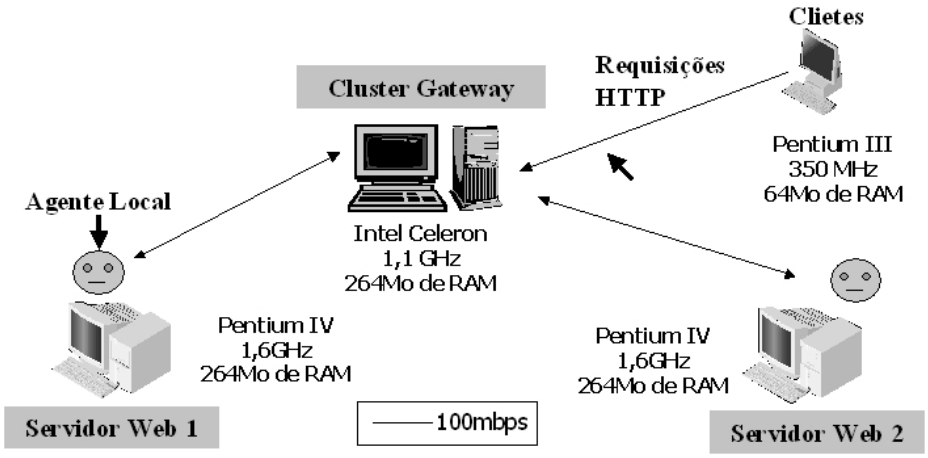



Figura 5 – Arquitetura de Hardware utilizada no experimento

No servidor Intel Celeron 1,1 GHz foi instalado o "Cluster Gateway" (CRM e HRG). Um MA foi instalado em cada servidor web. O Tomcat 5 foi utilizado como "container" para o serviço WEB.

### 6.2 – Descrição dos Experimentos e Resultados

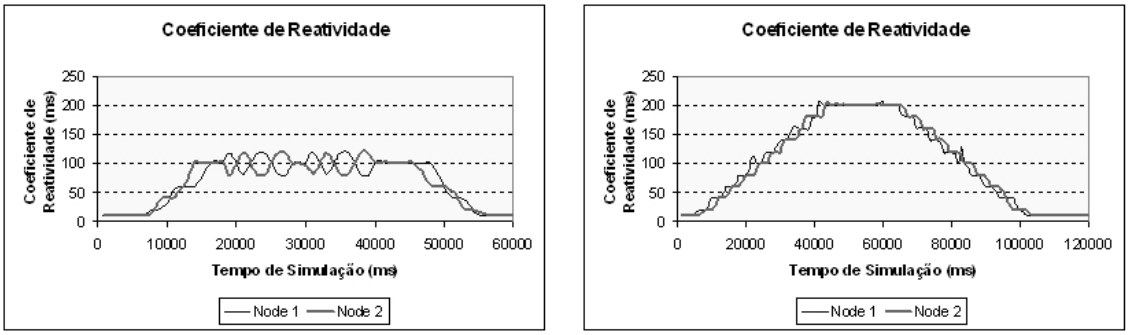
Nestes experimentos, o coeficiente de reatividade (carga) dos servidores Web foram monitorados durante o processamento de requisições HTTP.

No primeiro experimento (figure 6.a), foi instalado um serviço Web em ambos os servidores. Cada servidor gasta em media 10.000ms para processar o serviço em uma condição ideal (quando o servidor Web tem uma carga mínima). No equipamento Cliente foi instalado um objeto Java que gera 40 requisições HTTP para o serviço

instalado em intervalos de 1.000ms. Neste experimento a carga máxima medida no servidor Web 1 foi de 120ms e no servidor 2 de 121ms. A carga média medida foi de 60ms em ambos. O mesmo experimento foi realizado com um só servidor e teve uma carga máxima de 318ms e uma carga média de 134ms.

No segundo experimento (Figura 6.b), um serviço Web foi instalado em cada servidor que gasta em media 60.000ms para ser processado em uma condição ideal. O cliente gera 20 requisições HTTP para o serviço em intervalos de 2.000ms. Neste experimento a carga máxima medida foi de 206ms no servidor Web 1 e de 204ms no servidor Web 2 e a carga média medida em ambos foi de 102ms. O mesmo experimento foi realizado com um só servidor e teve uma carga máxima medida de 417ms e uma carga média de 216ms.

Na Figura 6 (a e b) pode-se observar que em ambos os casos a carga imposta pelas requisições do cliente foram proporcionalmente distribuídas entre os servidores Web disponíveis.




**(a)** **(b)**

Figura 6 – Distribuição da Carga entre os Servidores de um Cluster

## 6.3 – Experimentos de Diferenciação de Serviços

Nesta seção é apresentado um experimento simples que mostra a capacidade de controle de admissão e diferenciação de serviços da plataforma.

Nestes experimentos a classe de serviços "Gold" foi definida com um coeficiente de reatividade máximo de 40ms, a classe "Silver" com 80ms e a "Bronze" com 160ms. A classe "Best Effort" não tem nenhuma limitação. Neste experimento cada cluster possui dois servidores Web. A configuração de hardware utilizada para cada cluster é a mesma apresentada nos experimentos da seção 6.1. O CS (Class Scheduler) e os CGs (Cluster Gateways) foram instalados em um mesmo servidor (o notebook Intel Celeron).

Um serviço Web foi instalado em cada servidor. Cada servidor gasta em média 60.000ms para processar o serviço em uma condição ideal. Para cada classe de serviço, um cliente gera 20 requisições HTTP para o serviço em intervalos de 2.000ms.

Na Figura 7, pode-se observar o coeficiente de reatividade medido em cada cluster. Como pode ser visto, a plataforma foi capaz de manter o coeficiente de reatividade de cada classe próximo ao valor máximo especificado para a definição da classe. Quando o coeficiente de reatividade atinge o máximo especificado para o cluster de uma classe o CS passa a recusar o processamento de novas requisições daquela classe de serviços.

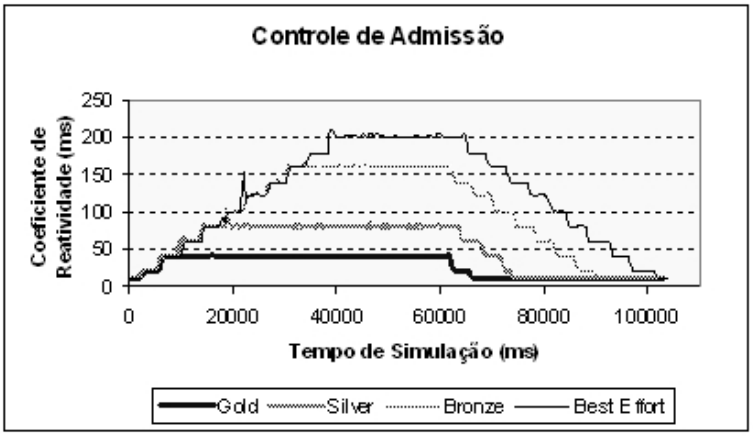



Figura 7 – Coeficiente de Reatividade por Classe de Serviço

## 7 – Conclusão

Este artigo apresentou uma plataforma para o balanceamento de cargas que oferece diferentes níveis de QoS para serviços Web. Serviços Web e objetos distribuídos coexistem na plataforma. Mecanismos de balanceamento de cargas e controle de admissão de requisições são transparentes para o provedor de serviços e para os desenvolvedores de aplicações. O nível de QoS oferecido ao cliente é medido através do "coeficiente de reatividade" de cada servidor que é diretamente relacionado com o tempo de resposta do cliente. Na plataforma é possível fazer a diferenciação de classes de serviços e a associação de um "coeficiente de reatividade" para cada classe. Também foram apresentados experimentos que mostram a capacidade da plataforma de balancear as cargas entre os servidores dos clusters e assegurar um nível específico de QoS para classes de serviços distintas.

Como trabalho futuro pretende-se: (1) utilizar uma técnica de predição de carga para reduzir o número de requisições rejeitadas pela plataforma; (2) implementar uma cooperação entre os agentes de monitoramento para realocar recursos entre os clusters quando necessário; (3) desenvolver mecanismos para permitir a renegociação da QoS entre os clientes e a plataforma e (4) Introduzir mecanismos de gestão de falhas.

Argumenta-se que a plataforma apresentada é uma boa solução para o aumento da QoS em aplicações na Internet através do uso eficaz dos recursos de hardware disponíveis de uma forma transparente.

## References

[1] Turgut, D.; Turgut, B.; Das, S.K.; Elmasri, R.; *Balancing loads in mobile ad hoc networks* Telecommunications, 2003. ICT 2003. 10th International Conference on , Volume: 1 , Page(s): 490-495 2003

[2] Marron, P.J.; *Design of a middleware service for scalable wide area network applications,* Distributed Objects and Applications, 2000. Proceedings. DOA '00. International Symposium, Page(s): 263-272 2000

[3] Rajagopalan, A.; Hariri, S.; *An agent based dynamic load balancing system* Autonomous Decentralized Systems, 2000. Proceedings. 2000 International Workshop Page(s): 164 –171 2000

[4] Zhou Yi; Ahmad, H.F.; Arfaoui, H.; Mori, K.; *Autonomous information allocation through mobile agents to achieve load balancing in distributed information service system,* Autonomous Decentralized System, 2002. The 2nd International Workshop on , Nov. 6-7 Page(s): 35 -41 2002

[5] P.M. Melliar-Smith, L.E. Moser, V. Kalogeraki, P. Narasimhan *Realize: Resouce Management for Soft Real-Time Distributed Systems* DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings , Volume: 1 , 25-27 Jan. Page(s): 281 -293 vol.1 2000

[6] Lap-Sun Cheung; Yu-Kwong Kwok; *The design and performance of an intelligent Jini load balancing service,* Parallel Processing Workshops, 2001. International Conference, Page(s): 361 - 366, 2001

[7] Damiani, E.; *An intelligent load distribution system for CORBA-compliant distributed environments,* Fuzzy Systems Conference Proceedings, 1999. FUZZ-IEEE '99. 1999 IEEE International , Volume: 1 Page(s): 332 -336 vol.1 1999

[8] Lu Lina; Liu Longguo; Yang Xinyu; *An agent-based load balancing mechanism: PLRM using Java* Technology of Object-Oriented Languages and Systems, 2000. TOOLS - Asia 2000. Proceedings. 36th International Conference on , 30 Oct.-4 Nov. Page(s): 176 -181 2000

[9] Desic, S.; Huljenic, D.; *Agents based load balancing with component distribution capability* Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002 Page(s): 327-331 2002

[10] Obeloer, W.; Grewe, C.; Pals, H.; *Load management with mobile agents* Euromicro Conference, 1998. Proceedings. 24th , Volume: 2 , 25-27 Aug 1998 Page(s): 1005 - 1012 vol.2 1998

[11] Cheung, Lap-Sun; Kwok, Yu-Kwong; *The design and performance of an intelligent Jini load balancing service,* Parallel Processing Workshops, 2001. International Conference, Page(s): 361 - 366, 2001

[12] JTC1/SC21/WG1. ISSO/IEC DIS 13236. *"Qualite of Service Framework"* – Outline, Febuary, 1996.

[13] R. Olejnik, A. Bouchi, B. Toursel ; *An Object observation for a Java Adaptative Distributed Application platform*, Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC'02) - 2002.

[14] Java Remote Method Invocation http://java.sun.com/products/jdk/rmi/

[15] The Apache Jakarta Project http://jakarta.apache.org/tomcat/index.html

[16] A. Serra; WS-DSAC – *An Dynamic DiffServ-based Load-Balancing and Admission Control Mechanism for Web Server Clusters*, Tecnical Report – 2004.