Uma Solução para Aumentar a Disponibilidade de Serviços em Clusters de Servidores Web Baseada em Sistemas Multiagentes

Carla Marques[1], Isabel Marinho[2], Giovanni Barroso[2], Mário Fiallos[2], Antônio Serra[3]

[1]Universidade do Estado do Rio Grande do Norte (UERN)
BR 110 Km 46,59600-900,Mossoró-RN-Brasil

[2]Universidade Federal do Ceará(UFC)
Campus do Pici, Fortaleza-CE 60455 760 - Brasil

[3]Centro Federal de Educação Tecnológica do Ceará
Av. 13 de Maio, 2081, Fátima, Fortaleza - CE 60040-531 - Brasil

{carla.katarina,isabelregio}@gmail.com,gcb@fisica.ufc.br,

fiallos@oi.com.br,prof.serra@gmail.com

Abstract. *In this paper is proposed a solution to increase the availability of services in clusters of web servers using Multi-agent system. The architecture, a formal specification in Petri nets and experimental results of the implementation of the solution are presented.*

Resumo. *Neste artigo é proposta uma solução para aumentar a disponibilidade de serviços em clusters de servidores web baseada em sistemas multiagentes. São apresentadas a arquitetura, a especificação formal em redes de Petri e resultados experimentais da implementação da solução.*

1. Introdução

Em algumas situações o aumento na demanda de serviços dos servidores Web resulta numa queda inaceitável na disponibilidade do sistema. Uma solução atraente para contornar este problema é a utilização de clusters de servidores web que oferecem maior poder de processamento para manter a disponibilidade do sistema em limites aceitáveis, mesmo com aumento significativo da carga de serviços.

Em clusters de servidores web, o atendimento das requisições é feito por um escalonador que recebe estas últimas e as envia ao servidor escolhido de acordo com a estratégia de escalonamento. Esta estratégia visa obter um maior desempenho e tem vários aspectos críticos a serem tratados, dentre os quais, o balanceamento de carga entre servidores.

Esse trabalho apresenta uma solução de reconfiguração dinâmica de clusters de servidores web baseada em sistemas multiagentes, incorporada à plataforma WS-DSAC [Serra et al. 2005], que visa minimizar a tarefa do administrador de clusters. As vantagens de se utilizar agentes se dá no fato que um sistema multiagente [Wooldridge 2002] é uma boa forma de prover o gerenciamento dinâmico dos recursos já que esta é uma solução inerentemente distribuída sem diminuir a performance da plataforma WS-DSAC.

As principais contribuições deste trabalho são: Realização de uma auto-reconfiguração de clusters, sem necessidade da interferência do administrador. Os agentes tomam decisões de realocação dinâmica de servidores baseado em informações coletadas através da interação com a plataforma WS-DSAC; Verificação da quantidade adequada de servidores a serem realocados entre os clusters, visto que o sistema toma suas próprias decisões de reconficonguração.

Este artigo está assim formatado: Uma descrição dos trabalhos relacionados é apresentado na seção 2. Na seção 3 são descritos aspectos relevantes da plataforma WS-DSAC, sobre a qual é implementado o sistema multiagente. A seção 4 apresenta o sistema multiagente e sua integração com o sistema WS-DSAC. A Especificação formal em Redes de Petri Coloridas da solução é apresentada na seção 5. Na seção 6 são mostrados resultados de simulação e na seção 7 são apresentadas as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Os clusters de servidores web e aspectos inerentes como por exemplo, balanceamento de carga, tem sido objeto de muitos estudos em computação.

Em [Serra et al. 2005] é apresentada uma plataforma que realiza o balanceamento de carga em Servidores Web baseada na diferenciação de serviços. Os servidores são agrupados em diferentes clusters Web de acordo com as classes de serviços estabelecidas. Cada cluster Web é responsável por atender uma classe específica de serviços e por garantir uma QoS (Qualidade de serviço) medida por meio do ”coeficiente de reatividade”(reactivity coefficient - RC) [Olejnik et al. 2002][1],que mede sua carga estimando o tempo que uma tarefa espera para utilizar a CPU.

Vários trabalhos utilizam a tecnologia agente para realizar balanceamento de carga dinâmico [Wang et al. 2007, Herrero et al. 2007, Nehra and Patel 2007]. A principal limitação desses trabalhos é que estes não realizam uma reconfiguraçao dinâmica de clusters (i.e., eles não alocam/desalocam servidores dinâmicamente aos clusters). Dessa forma, estes trabalhos requerem a presença de um administrador para administrar as constantes modificações que ocorrem em um ambiente distribuído. Este trabalho manual é cansativo e passível de falhas, especialmente quando a configuração muda constantemente devido ao crescimento do cluster ou a novas necessidades de recursos.

Alguns trabalhos apresentam soluções que realizam reconfiguração dinâmica de clusters. Em, [Zhang et al. 2006] é o trabalho mais semelhante ao que estamos propondo já que também realiza reconfiguração dinâmica de clusters utilizando agentes. Este trabalho possibilita um gerenciamento fácil e confiável para clusters. O principal problema deste trabalho é que o gerenciamento é feito ao nível do Sistema Operacional (chamado Fire Phoenix). Apesar deste sistema operacional coexistir com outros sistemas operacionais disponíveis no mercado ter dois diferentes kernels numa mesma máquina gera um overhead extra. No trabalho apresentado em [Sung et al. 2007], é proposto um mecanismo de auto-configuração utilizando a tecnologia agentes. Este mecanismo permite a alocação dinâmica dos recursos disponíveis em um cluster sem a necessidade de intervenção humana. Porém quem gerencia estas tarefas é um servidor central, o qual é um ponto de falha.

[1]Esta técnica mede o tempo de espera de uma tarefa para uso dos recursos da CPU.

Um outro exemplo, [Adam and Stadler 2005] apresenta uma solução para alocar servidores em um cluster para o processamento de requisições e ativar automaticamente um outro servidor em standby quando a carga do cluster aumente. Inicialmente, uma requisição é alocada para um servidor randomicamente. Se este servidor não puder processar a requisição, esta é então enviada para outro servidor, e assim por diante, até que um servidor seja capaz de processá-la ou o tempo máximo para atendimento de uma requisição seja esgotado. Esta abordagem baseada em redirecionamento apresentada pode ser ineficiente. Além do mais, nenhum balanceamento de cargas é realizado e apenas um cluster foi considerado. No sistema multiagente proposto nesse trabalho os agentes aprendem para realizar a melhor reconfiguração no futuro.

A solução apresentada neste artigo visa realizar uma reconfiguração dinâmica de clusters de servidores web utilizando sistemas multiagentes minimizando a tarefa do administrador de clusters.

3. Visão Geral das Plataformas

Nesta seção é apresentada uma visão geral das plataformas WS-DSAC e Multiagente.

3.1. Plataforma WS-DSAC

O WS-DSAC é uma plataforma que tem como objetivos principais: balancear a carga imposta pelas requisições atendidas, garantir a QoS estabelecida para cada classe de serviços e utilizar de forma eficaz os recursos de processamento disponíveis. A plataforma WS-DSAC (Figura 1) é composta pelos seguintes 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, utilizando a plataforma WS-DSAC, envia cada requisição para um ”Cluster Gateway”específico. O ”Cluster Gateway”escolhe o servidor Web menos carregado para processar a requisição enviada pelo ”Class Switch”. A

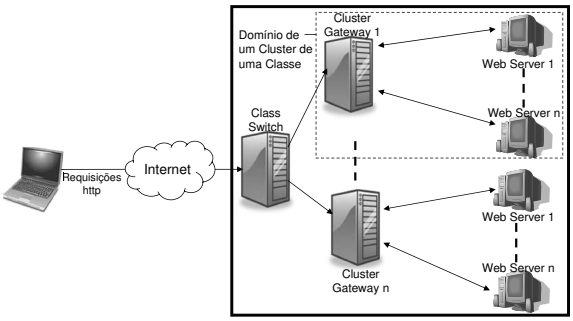

Figura 1. Visao Geral da Plataforma WS-DSAC.

plataforma oferece diferentes níveis de QoS, um nível diferente para cada classe de serviço. Esses serviços são instalados em um dado número de servidores Web e podem ser compostos por serviços Web e objetos distribuídos. Requisições que chegam podem pertencer a diferentes classes de serviço. O administrador da plataforma associa a cada classe de serviço uma carga máxima que pode ser atingida pelo seu ”class cluster”. Um parâmetro de QoS denominado ”Coeficiente de Reatividade”é associado a cada classe de serviços pré-estabelecida e este parâmetro é usado em cada servidor Web.

A estratégia adotada em [Serra et al. 2005] para a realocação de recursos entre as classes de serviços se baseia em uma mudança no modo de funcionamento de cada ”class

cluster". Em um determinado intervalo de tempo, cada cluster pode estar em um de três modos possíveis: "modo compartilhado"; "modo exclusivo"ou "modo saturado". Quando está no "modo compartilhado"o cluster da respectiva classe possui recursos disponíveis que podem ser utilizados por outras classes de serviços sem comprometer o contrato estabelecido com a classe "nativa"do cluster durante um intervalo de tempo pré-definido. Estando neste modo de trabalho o "class cluster"pode atender requisições de diferentes tipos de classes. Quando o cluster passa ao "modo exclusivo", ele só aceita requisições da classe "nativa". Quando o servidor passa a esse modo, significa que os níveis de carga chegaram a um patamar onde, aceitar requisições de outras classes, pode implicar na rejeição de sessões da classe nativa.

Quando o cluster passa ao "modo saturado"ele não aceita nenhuma nova requisição. A mudança de modo de funcionamento é baseada em dois parâmetros dinâmicos do cluster (recalculados a cada intervalo de tempo), R_{emk} e k, e dois estáticos que são R_{max} e R_{ac}, onde: R_{emk} é o valor limite do RC que pode ser alcançado pelo cluster de uma classe, e a partir deste valor, o cluster passa a trabalhar em "modo exclusivo", enquanto o RC está abaixo deste valor ele trabalha em "modo compartilhado"; k representa a carga estimada do cluster para o período de tempo seguinte e determina uma margem de segurança para que os contratos de QoS estabelecidos com a classe nativa sejam respeitados; R_{max} contém o valor máximo que pode atingir o R_{emk}; R_{ac} é o limite estabelecido para o cluster trabalhar no modo exclusivo.

Quando uma requisição chega à plataforma o "class switch"identifica qual é a sua classe. Dado que a requisição pertence à classe "i"e que o cluster "m"é o menos carregado, o "class switch"atua da seguinte forma: se o valor de ki for menor ou igual ao valor de R_{max} o cluster trabalha no modo compartilhado; se o valor de ki alcançar o valor de R_{max}, o cluster modifica o seu modo de trabalho para exclusivo; se o valor de ki alcançar o valor de R_{ac}, o cluster modifica o seu modo de trabalho para saturado e passa a não mais atender a nenhuma requisição.

Como pode ser observado pelo esposto acima, as variáveis R_{emk}, R_{max}, R_{ac} e k, apresentadas anteriormente, controlam o funcionamento da plataforma WS-DSAC. A partir desses parâmetros foi elaborada uma solução com a utilização de multiagentes, para monitoramento dos mesmos e com base nestas informações serem tomadas decisões, com o objetivo de gerar uma melhor disponibilidade de recursos do sistema. Esta nova arquitetura e um novo mecanismo (Realocação Dinâmica de Servidores em Clusters Web - RDSC) serão detalhados na seção 3.2.

3.2. Plataforma Multiagentes

O principal objetivo do mecanismo de realocação dinâmica apresentada neste artigo é gerar uma melhor disponibilidade de recursos do sistema de forma dinâmica, ou seja, o sistema deve ser capaz de interagir com a plataforma WS-DSAC, aprender com ela e gerar informações de alerta ou sugerir alterações para o administrador, que pode ser, por exemplo, aumentar servidores para uma classe de serviços que esteja necessitando de recursos. Complementando esse mecanismo, um sistema multiagente é uma boa forma de prover o gerenciamento dinâmico dos recursos já que esta é uma solução inerentemente distribuída. Desta forma, a realocação dinâmica baseada em sistemas multiagente é um mecanismo para auxiliar o trabalho do administrador do sistema, evitando assim sua

intervenção e acompanhamento constantes, como também auxiliá-lo na decisão, por uma redefinição na estratégia de distribuição dos recursos entre os diversos clusters.

A nova arquitetura apresentada na seção 2 que, será detalhada na seção 4, bem como a interação de seus agentes com os módulos da plataforma WS-DSAC.

4. A Arquitetura Multiagente Proposta

A nova arquitetura (Figura 2) é constituída da plataforma WS-DSAC (Figura 1), agregada ao módulo de Realocação Dinâmica de Servidores em Clusters (RDSC). Este módulo assume o papel de coletar e produzir informações estatísticas (taxa de requisições recusadas, média do coeficiente de reatividade por classe de serviços, etc.) sobre todo o sistema, como também fazer alterações na configuração dos clusters, se necessário. Os agentes se comunicam através de troca de mensagens remotas. As ações tomadas pelos agentes são baseadas em consultas a uma base de regras, que contém informações de controle do sistema como limite de capacidade máxima de carga para cada cluster, configuração dos clusters, porcentagens de carga monitoradas, etc.

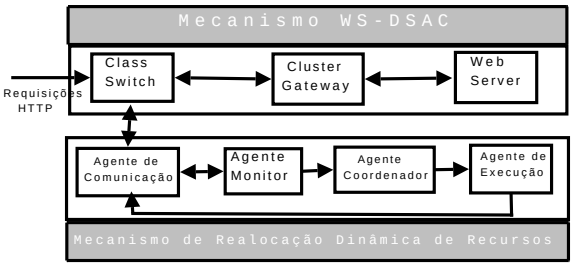

Figura 2. Comunicação entre os elementos do Mecanismo de Realocação Dinâmica de Servidores em Clusters.

4.1. O Mecanismo RDSC

O RDSC (Figura 2), é composto por quatro módulos e as suas funções são descritas a seguir:

O Agente de Comunicação faz a comunicação com o WS-DSAC, recebendo e enviando informações de parâmetros dos clusters e armazenado-as.

O agente da Camada de Monitoramento, instalados nos servidores Web, fica coletando informações do estado de cada servidor durante intervalos específicos de tempo. Esses intervalos entre os monitoramentos permitirão analisar o comportamento real da carga na plataforma WS-DSAC para tomada de decisão por parte do sistema RDSC. As informações de carga dos clusters coletadas pelo Agente de Monitoramento também serve para ativar as informações de controle armazenadas na base de regras do Agente Coordenador. Estas

O Agente Coordenador realiza o planejamento da capacidade de recursos de processamento da plataforma. Este agente administra as interações entre os vários agentes e verifica a Base de Regras que contém as ações a serem tomadas pelos Agentes de Execução que são acionadas pelas informações de disparo enviadas pelo agente monitor. Estas informações são atribuídas a variáveis, tais como, requisições rejeitadas, carga dos servidores, etc. Quando ocorre alguma modificação nestas variáveis o Agente Coordenador envia um estímulo ao Agente de Execução específico.

Os Agentes daCamada de Execução executam tarefas específicas. Eles recebem as informações de disparo do Agente Coordenador e executam a tarefa para a qual foram projetados. Todas as variáveis utilizadas são descritas em [Serra et al. 2005].

4.2. Algoritmo RDSC

O mecanismo RDSC se propõe a realizar o gerenciamento de clusters de forma dinâmica, onde as informações são colhidas por agentes, conforme discutido na seção anterior, as quais irão alimentar a base de regras do sistema, geradas inicialmente pelo administrador e depois alimentada pelos próprios agentes. Estes agentes exercerão ações sobre o mecanismo a fim de melhorar a eficiência da utilização dos recursos disponíveis, sendo capazes de tomar decisões, de forma que o sistema de clusters se adapte dinamicamente às necessidades impostas pelo ambiente. A seqüência de funcionamento do RDSC é descrita no Algoritmo 1. Como mostrado no Algoritmo 1, o servidor menos carregado é redire-

```
While (chagam requisiçoes http){
    Mecanismo WS-DSAC executa;
    Agente Monitor  solicita informações ao mecanismo WS-DSAC das variáveis thresholds (Rac, pki e
        Remk);
    Agente de Comunicação solicita ao mecanismo WS-DSAC os valores dessas variáveis;
    Agente Monitor  recebe informações solicitadas;
    Se (ρk i <= 80%Rac ) então {
            Enviar mensagem de monitoramento ao Agente Coordenador;
            Agente Coordenador envia informação de controle ao Agente de Execução Carga_Máxima;
            Agente de Execução solicita ao Agente de Comunicação informações de less loaded cluster e
              less loaded servidor pertencente a este cluster.;
            Agente de execuçao envia menasagem de execuçao ao Agente de Comunicaçao que
              redireciona para o cluster que está prestes a saturar o servidor menos carregado. }
        }
```

Algoritmo 1

cionado para o cluster que está prestes a saturar, possibilitando assim, que este cluster possa receber mais recursos e consiga atender prioritariamente as suas requisições nativas e com isto a quantidade de requisições rejeitadas poderá diminuir.

Com o objetivo de validar essa nova arquitetura apresentada, foi realizada a sua modelagem em Redes de Petri Coloridas [Jensen 1996] e, a partir desta, foram realizadas simulações. A modelagem, bem como as simulações, serão apresentadas na seção 5.

5. Modelagem da Arquitetura em Redes de Petri Coloridas

Nesta seção é apresentada a modelagem da arquitetura do modelo de realocação dinâmica de recursos descrita na seção 4 e ilustrada na Figura 2. A modelagem foi feita em redes de Petri Coloridas, utilizando-se a ferramenta CPNTools. É apresentada na Figura 3, a

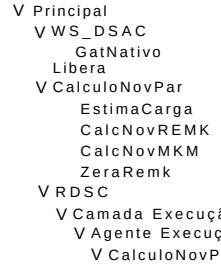

Figura 3. Hierarquia de Páginas da Modelagem em Redes de Petri Coloridas.

visão hierárquica das páginas e subpáginas que compõem o modelo. A página Principal representa o funcionamento geral da nova arquitetura detalhada na Figura 2 (WS-DSAC/RDSC), e é através dela que são executadas as outras subpáginas: WS-DSAC, Libera, CalculoNovPar e RDSC. A subpágina WS-DSAC modela o balanceamento de

cargas e a diferenciação de serviços; ela é composta por uma outra subpágina, GatNativo, onde são modeladas as alterações de carga de um servidor. A subpágina Libera modela a liberação de requisições e cargas dos servidores, após o atendimento das requisições. A subpágina CalculoNovPar possui a função de atualização dos parâmetros utilizados pela plataforma WS-DSAC como, por exemplo, o cálculo de novas cargas estimadas para os servidores pertencentes aos clusters, entre outros, e que é representada pela subpágina EstimaCarga. Finalmente, na página RDSC é modelado o mecanismo RDSC; ela também é composta por outras subpáginas que modelam os agentes que compõem o mecanismo como, por exemplo, o Agente de Execução, que é representado pela subpágina Agente Execução.

Na Figura 4 é apresentada a página Principal onde aparecem as transições de substituição (retângulos duplicados) representando as outras subpáginas apresentadas na página hierárquica (Figura 3). Por exemplo, a transição WS-DSAC representa a rede WS-DSAC, que modela o funcionamento do mecanismo em si. A modelagem foi feita

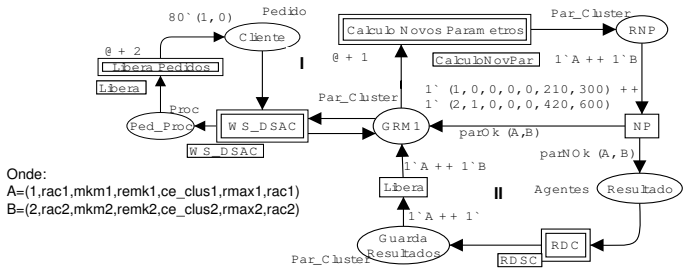

Figura 4. Página Principal da Modelagem em Redes de Petri Coloridas.

em duas etapas: a primeira modelando a plataforma WS-DSAC (rede I) e a segunda após a inclusão do mecanismo RDSC (rede II).

A rede da Figura 4 modela o seguinte: quando um pedido chega, a plataforma WS-DSAC é executada, e logo após o processamento, o pedido é liberado, bem como a carga do servidor que o atendeu. Em intervalos de tempo pré-definidos, a transição de substituição CalculoNovPar (Figura 5) é ativada, atualizando os parâmetros usados pela plataforma WS-DSAC. Após cada execução desta transição (CalculoNovPar), ou seja, após cada atualização dos parâmetros utilizados pelo WS-DSAC, é realizada uma verificação, da necessidade ou não, de ser feita uma realocação de recursos no sistema. Se a resposta for positiva, então a transição de substituição RDSC é executada, caso contrário os parâmetros do WS-DSAC, que foram atualizados, voltam novamente para serem usados pela rede e o processamento de pedidos continua.

Na Figura 6 é apresentada a rede que detalha a transição de substituição RDSC (4). Esta transição será executada se a carga do cluster em questão não estiver dentro do limite estabelecido para a sua classe nativa, definido pelo administrador. Neste caso, o agente de monitoramento enviará a informação de disparo ao agente coordenador e este ativará o agente de execução (transição Camada Execução). Após a execução desta transição os cálculos dos parâmetros usados pelo WS-DSAC são refeitos, visto que recursos foram realocados e a configuração dos clusters provavelmente mudou, justificando assim uma nova chamada dessa transição (CalculoNovPar) dentro daquela (RDSC).

A partir do modelo apresentado, foram realizadas várias simulações com diferentes cenários de funcionamento do sistema, para validação do mecanismo RDSC em

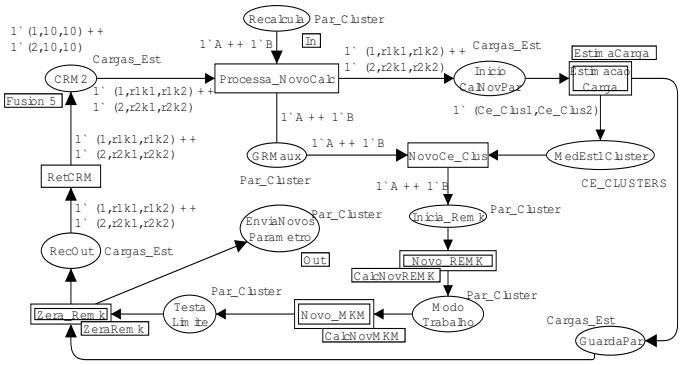

Figura 5. Página CalculoNovPar.

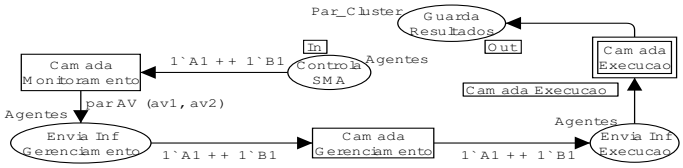

Figura 6. Página RDSC

garantir a QoS contratada para cada classe de serviços em momento de utilização crítica dos recursos. As simulações foram realizadas utilizando-se a ferramenta CPNTools e serão apresentadas a seguir.

6. Experimentos

As simulações foram realizadas em duas etapas: a primeira foi realizada sem realocação dinâmica de clusters, ou seja, foi simulado apenas a plataforma WS-DSAC usando apenas a rede I (Figura 4). Já na segunda etapa, o mecanismo RDSC foi adicionado ao WS-DSAC, e a rede II (Figura 4) foi adicionada à rede I, da mesma figura.

Para as simulações, dois domínios diferentes de classes de pedidos foram definidos na plataforma (classes 0 e 1) cujos parâmetros são descritos na seção 3. Também foi usada na simulação uma arquitetura contendo dois clusters (clusters 0 e 1), cada um possuindo dois servidores, sendo atribuído a cada cluster uma classe nativa de serviços a serem atendidos, ficando o cluster 0 associado à classe 0 e o cluster 1 à classe 1. Na simulação usando o RDSC, para representar o aumento de recursos em um cluster, ou seja, a retirada de um servidor de um cluster e a sua adição ao outro cluster, aumentaram-se os parâmetros de uma classe de serviços, na mesma proporção que reduziram-se os parâmetros da outra. As variáveis R_{ac} e R_{max} têm os valores 300 e 210 respectivamente para o cluster 0. Para o cluster 1, os valores são respectivamente 600 e 420.

Os resultados de simulação são comparados e apresentados nas seções seguintes.

6.1. Momentos Críticos

No primeiro experimento foi simulada uma situação em que, um cliente envia requisições pertencentes à classe ”0”(uma a cada 1s) ao mesmo tempo em que um outro cliente envia requisições pertencentes à classe ”1”(uma a cada 1s), sendo as quantidades de requisições iguais para as duas classes. Este experimento foi simulado primeiro sem realocação dinâmica (Figura 7), isto é, só com o WS-DSAC, e depois foi feita simulação

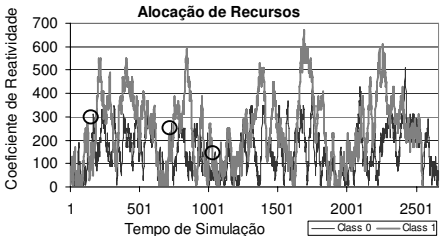

Figura 7. Alocação de recursos com carga nativa das duas classes sem RDSC.

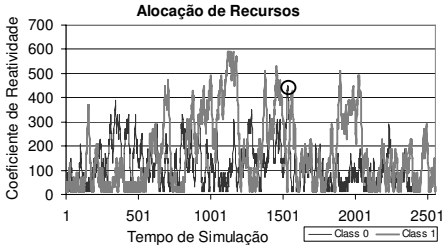

Figura 8. Alocação de recursos com carga nativa das duas classes com RDSC.

com realocação dinâmica de servidores nos clusters (Figura 8), ou seja, com a introdução do mecanismo RDSC.

A Figura 7 mostra a distribuição de cargas entre os dois clusters. As simulações mostraram que a partir do tempo de simulação, t=168 (este instante é sinalizado com o primeiro círculo da esquerda para direita) ocorrem as primeiras rejeições da classe 0 (RC>300), que se repetem durante toda a simulação. As simulações mostraram também o comportamento dos clusters ao longo da simulação, p.e., em t=785, o cluster 0 fica sobrecarregado (RC=210), mudando seu modo de funcionamento para o modo exclusivo. O cluster 1 por sua vez passa a atender as requisições em modo compartilhado, RC<420. Esta distribuição de cargas não impede o sistema de rejeitar requisições, pois algumas vezes os clusters ultrapassaram os seus limites máximos de regime compartilhado. Esta situação é sinalizada na Figura 7 pelo segundo círculo. O terceiro círculo mostra uma situação com RCs que permitem o funcionamento dos clusters em modo compartilhado. Já na Figura 8, com realocação dinâmica de servidores, as simulações mostraram que, apesar de serem enviadas quantidades iguais de requisições pertencentes às duas classes, o cluster 1 não saturou e o cluster 0 chegou em menor número de vezes aos seus limites de carga. Comparando-se com a simulação da Figura 7, o cluster 1 se manteve sempre com o RC abaixo de 600, ao contrário do que aconteceu na Figura 7, onde ele ultrapassa este limite causando uma maior taxa de rejeição de requisições, e onde também o cluster 0 obteve o seu ponto máximo (RC=510) contra (RC=450) (ver círculo na Figura 8). Conclui-se, portanto, que a utilização do RDSC, provocou uma realocação dinâmica de recursos entre os dois clusters, e a conseqüentemente diminuição do índice de saturação nos mesmos.

Em ambos os casos (Figuras 7 e 8), foram feitas várias simulações com o objetivo de calcular a taxa média de rejeições, sendo observado que, no primeiro caso, a taxa média de rejeição das requisições foi de 3,93% enquanto que para o segundo foi de 0,04%, comprovando um melhor desempenho do sistema.

6.2. Momentos de Saturação

Foram feitas simulações onde só são enviadas requisições pertencentes à classe 0 (Figuras 9, 10, 11 e 12). Desta forma, o cluster pertencente à classe 0 satura mais rapidamente, pois ele possui os menores limites de carga. O gráfico da Figura 9 refere-se à simulação sem o uso do mecanismo RDSC, ao contrário das Figuras 10, 11 e 12 onde o RDSC foi aplicado. É importante citar que o percentual monitorado de carga dos clusters, pelos agentes, foi modificado em vários níveis, sendo de 80% no primeiro caso (Figura 10), 85% no segundo (Figura 11) e 90% no último (Figura 12) e que em todos os três casos foram deslocados, quando necessários, 10% de recursos do cluster 1 para o cluster 0. Isto quer dizer que, por

exemplo, se referindo à simulação da Figura 10, quando a carga do cluster 0 atingiu 80% da sua carga limite, 10% dos recursos do cluster 1 foram deslocados para ele. Pode-se

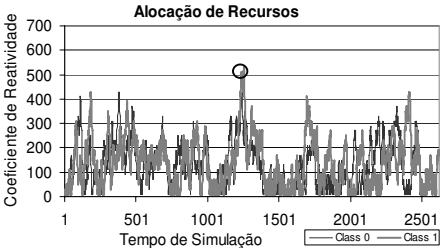

Figura 9. Alocação de recursos com carga nativa da classe 0 sem RDSC.

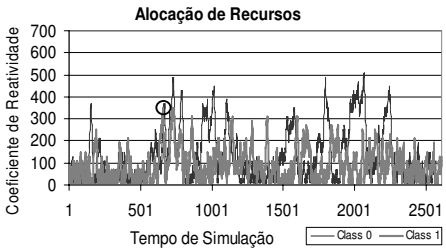

Figura 10. Alocação de recursos com carga nativa da classe 0 sem RDSC.

observar também que no caso da Figura 9), apesar de haver distribuição de cargas, ocorre rejeição de requisições em vários momentos devido à saturação do cluster 0 (RC>300). Já na Figura 10, observa-se que com a realocação dinâmica de recursos, o cluster 1, apesar de ceder recursos para o cluster 0, não saturou e manteve o seu nível de carga médio mais baixo que o da Figura 9. Isto pode ser comprovado através da verificação dos pontos de carga máxima do cluster 1 nas duas simulações citadas. Na Figura 9 observamos que o cluster 1 atinge um valor máximo de carga (RC=530) em t=1254, e na Figura 10 o seu ponto máximo de carga (RC=350) ocorreu no instante 656 e que o mesmo trabalhou o restante do tempo de simulação sempre abaixo disso. Dois instantes ou momentos de saturação do cluster 0 são representados por círculos nas Figuras 9 e 10.

Como nas duas situações anteriores (Figuras 7 e 8) várias simulações foram feitas com o objetivo de se calcular as taxas de rejeição, e para este experimento constatou-se que 9,8% das requisições enviadas foram rejeitadas na rede da Figura 9, e 0% foi rejeitada para a rede da Figura 10, ou seja , mesmo cedendo recursos para o cluster 0, o cluster 1 não saturou. Dessa forma, o sistema conseguiu atender a todas as requisições enviadas, melhorando a utilização dos recursos.

Na simulação relativa à Figuras 11 o agente de monitoramento notifica o agente coordenador quando a carga do cluster 0 atingir 85% do seu limite e na Figura 12 quando atingir 90%.

A taxa de rejeição observada para a rede da Figura 11 foi de 0,02% e a da rede da Figura 12 foi de 0,05%, indicando que à medida que se aumenta o percentual de monitoramento do limite de carga do cluster, a taxa de rejeição também aumenta. Isto indica que, se as atitudes a serem tomadas pelos agentes forem feitas com mais antecedência, pode-se obter um maior desempenho do sistema. É importante destacar que comparando

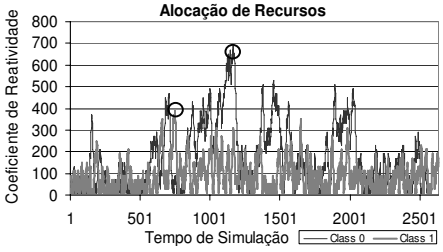

Figura 11. Alocação de recursos com RDSC e ação do agente com 85% da carga.

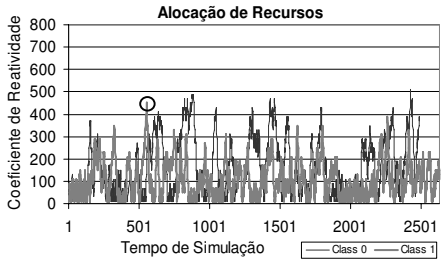

Figura 12. Alocação de recursos com RDSC e ação do agente com 90% da carga.

o comportamento das simulações representadas nas Figuras 10, 11 e 12 têm-se:

Na Figura 10 temos 13 picos de carga máxima (RC $>=$ Rmax) referente ao cluster 0, contra 15 do mesmo cluster na Figura 11, e 17 na Figura 12 indicando que quando a realocação de recurso é feita com 80% do limite de carga do cluster, o sistema necessita de menos reajustes do que quando as atitudes necessárias são tomadas quando o cluster tem cargas maiores (85% e 90%);

Na Figura 10 o cluster 1 se manteve sempre com nível de carga baixo, a maior parte da simulação com RC$<$300, e, consequentemente, se obteve o menor tempo de resposta;

Na simulação da Figura 11, os clusters se comportaram mais irregularmente do que na Figura 10, alcançando valores de carga próximos dos limites estabelecidos. De fato, as simulações mostraram esta situação nas coordenadas (1164, 670) para o cluster 0, e no ponto (748, 390) para o cluster 1. Os círculos na figura denotam a localização destes instantes;

Na Figura 12, o cluster 1 atingiu um valor de carga que ultrapassou o seu limite em (555, 450), sinalizado pelo círculo. Como mostrado também em vários outros momentos da simulação seus valores foram bastante altos, mais inclusive do que nas outras simulações (Figuras 10 e 11).

Diante dos resultados apresentados, pode-se concluir que dentre as simulações apresentadas, a que se comportou melhor foi a que fez a realocação de recursos com monitoramento de 80% da carga do cluster 0 (Figura 10), se mantendo com maior coerência com os objetivos propostos neste trabalho, concluindo assim que a demora na tomada de atitudes para realocação de recursos nos clusters provoca uma sobrecarga nos seus servidores e consequentemente rejeição de requisições. Dessa forma a incorporação do RDSC ao WS-DSAC, realiza a realocação dinâmica dos recursos, diminuindo o número de rejeições, e conseqüentemente melhorando significativamente o desempenho do sistema.

7. Conclusões

Este artigo apresentou a concepção, a especificação e a modelagem de um protótipo de um sistema multiagente (SMA) para a realização de realocação dinâmica de recursos em clusters de servidores web no nível do gerenciamento de capacidade (capacity management). Foi idealizada uma arquitetura em camadas dos agentes, bem como suas funcionalidades. Foi realizada a modelagem desta arquitetura em redes de Petri coloridas, utilizando a ferramenta CPNTools. Finalmente, experimentos foram realizados em diferentes situações e constatou-se pelos gráficos que a arquitetura de realocação dinâmica proposta melhorou o tempo de resposta aos usuários, promoveu uma melhor utilização dos recursos (”fairness”) e principalmente, as taxas de rejeição foram reduzidas consideravelmente. Quando foi mudado o percentual da variável monitorada pelo agente, observando-se os limites de 80%, 85% e 90%, respectivamente, verificou-se que à medida que aumentava o percentual de monitoramento da carga máxima do cluster, a taxa de rejeição de requisições aumentava, concluindo-se assim que se for deixada para ser feita a realocação de recursos quando o cluster monitorado estiver muito perto dos limites estabelecidos, a taxa de rejeição aumentará.

É importante também ressaltar que, uma das principais vantagens deste mecanismo é que através dele, pode-se melhorar a utilização dos recursos existentes, sendo deslocados de um cluster para outro a fim de que todas as requisições sejam atendidas. No nosso caso, para os valores dos parâmetros iniciais usados em nossos experimentos, foi necessário que apenas 10% dos recursos de um cluster fossem redirecionados para o outro, para que a taxa de rejeição diminuísse e chegasse a 0. Essa porcentagem poderia ser diferente caso os valores dos parâmetros usados fossem outros. Também estamos iniciando a fase de testes da implementação do mecanismo RDSC associado ao WS-DSAC, o que possibilitará a realização de um conjunto de experimentos em uma plataforma real de testes. Assim pretendemos avaliar melhor a eficácia do sistema multiagente na realocação dinâmica dos recursos. Dando continuidade ao trabalho, serão realizados experimentos para análise do comportamento do sistema multiagente, baseados em variações reais de carga. Os resultados preliminares levam a acreditar na eficácia da solução proposta.

Referências

Adam, C. and Stadler, R. (2005). Adaptable server clusters with QoS objectives. *International Symposium on Integrated Network Management*, pages 149–162.

Herrero, P., Bosque, J. L., Salvadores, M., and Perez, M. S. (2007). An agents-based cooperative awareness model to cover load balancing delivery in grid environments. *Lecture Notes in Computer Science*, pages 64–74.

Jensen, K. (1996). *Colored Petri Nets, Basic Concepts, Analysis Methods and Practical Use*, volume 1, 2nd edition. Springer.

Nehra, N. and Patel, R. B. (2007). Towards dynamic load balancing in heterogeneous cluster using mobile agent. *International Conference on Computational Intelligence and Multimedia Applications*, pages 15–21.

Olejnik, R., Bouchi, A., and Toursel, B. (2002). An object observation for a java adaptative distributed application platform. *International Conference on Parallel Computing in Electrical Engineering*, pages 171–176.

Serra, A., Cardoso, K., Barroso, G., and Ramos, R. (2005). Controle de admissao e diferenciacao de servicos em clusters de servidores web. *Proceedings of the SBRC-SBC, 2005*.

Sung, H., Choi, B., Kim, H., Song, J., Han, S., Ang, C. W., Cheng, W. C., and Wong, K. S. (2007). Dynamic clustering model for high service availability. *International Symposium on Autonomous Decentralized Systems*, pages 311–317.

Wang, J., Ren, Y., Zheng, D., and Wu, Q. (2007). Agent based load balancing middleware for service-oriented applications. *International Conference on Computational Science, Part II*, pages 974–977.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John Willey and Sons Ltd.

Zhang, Z. H., Meng, D., Zhan, J. F., Wang, L., Wu, L. P., and Huang, W. (2006). Easy and reliable cluster management: The self-management experience of Fire Phoenix. *International Parallel and Distributed Processing Symposium*, page 8pp.