UFC

UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA
PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

UM ESTUDO SOBRE INTERRUPÇÃO E REINICIABILIDADE DE PROCESSOS EM CLUSTERS DE SERVIDORES WEB

PITÁGORAS GRAÇA MARTINS

FORTALEZA - CEARÁ
2006

UNIVERSIDADE FEDERAL DO CEARÁ

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

PITÁGORAS GRAÇA MARTINS

UM ESTUDO SOBRE INTERRUPÇÃO E REINICIABILIDADE DE PROCESSOS EM CLUSTERS DE SERVIDORES WEB

Dissertação apresentada à Universidade Federal do Ceará, como parte das exigências do Programa de Pós-Graduação em Engenharia de Teleinformática, Área de Concentração: Tecnologia, para obtenção do título de Mestre em Engenharia de Teleinformática

Orientador : Giovanni Cordeiro Barroso

Co-Orientador : Ronaldo Fernandes Ramos

FORTALEZA
Ceará - Brasil
Julho de 2006

RESUMO

WS-DSAC é uma plataforma desenvolvida sob um cluster de servidores Web com balanceamento de cargas capaz de realizar o controle de admissão e diferenciação de serviços para permitir a diferenciação da QoS oferecida aos clientes e utilizar de forma eficaz os recursos disponíveis.

Quando requisições em excesso são enviadas para o cluster, o sistema se torna "saturado". Isto significa que o mecanismo de controle de admissão não é mais capaz de manter o tempo de resposta necessário para que se possa garantir a qualidade de serviço do sistema como um todo. Neste trabalho é apresentado um mecanismo que utiliza o conceito de interrupção e reativação de processos para otimizar a capacidade de processamento do sistema com a finalidade de evitar o problema da "saturação". Este mecanismo interrompe apenas os processos que podem ocasionar sobrecarga do servidor e reinicializando-os posteriormente quando a carga do servidor volta a melhores condições de processamento.

Desta forma, é possível aumentar o número de requisições privilegiadas às expensas de outros serviços que podem esperar, diminuindo também número de requisições não atendidas.

ABSTRACT

With the expansion of services offered through the Internet and the popularization of the Web technology in the modern world, the toughest problems Web server administrators face are how to dimension infrastructure and how to manage the high workload during intense processing moments. As a result, we have websites with extremely high response times, due to bad resource management.

This paper presents a mechanism of high availability for services and process management, for the optimization of server capacity, through a technique called process interruption and restart.

The interruption and restart technique was employed to enable services to be available even through high workload moments. The restarts have three important advantages. First, it either makes a process go back to its initial state or become immune to problems. Second, the restarts have the property of increasing system reliability. The suspension and restart technique is utilized to control resource consumption under heavy processing workload. Another advantage is that restarting is a simple mechanism, easy to implement and retrieve.

This has created the possibility of creating a new layer that allows better process management under severe workloads. It provides better machine performance, lower response time, and greater service availability, in addition to meeting SLA needs – QoS contracts.

SUMÁRIO

LISTA DE TABELAS

LISTA DE ILUSTRAÇÕES E FIGURAS

1. Introdução

Esta dissertação apresenta um mecanismo de alta disponibilidade de serviços e de gerenciamento de processos para otimização da capacidade de servidores por meio de interrupção e reativação de processos.

Neste Capítulo será discutida a motivação para o desencadeamento do estudo desta dissertação, os objetivos a serem alcançados e a estruturação deste trabalho.

1.1 Problema e motivação

A cada dia os sistemas computacionais estão mais complexos. Isto fortalece a teoria da inevitabilidade da falha, ou seja, à medida que a complexidade dos sistemas aumenta, cresce também a certeza de que falhas necessariamente estarão presentes na concepção ou na operação de tais sistemas.

A suspensão não previsível de serviços é responsável por diversos prejuízos. Para que se tenha idéia disso, servidores que possuem um tempo de inatividade de 8 a 80 horas por ano, ou seja, servidores que apresentam uma disponibilidade que varia de 99.0% a 99,9%, podem causar prejuízos a empresas como a Amazon.com na ordem de $200.000 ou até $6.000.000 para uma agência corretora [Kembel 1998]. Torna-se de extrema importância proporcionar um serviço sem interrupções e que atenda também a quesitos de qualidade de serviço e também de segurança pelo fato da disponibilidade do serviço ser um fator crítico em determinadas circunstâncias.

Por isso a alta disponibilidade em serviços computacionais tem se tornado um ramo de estudo essencial não só do ponto vista científico, mas também comercial.

Procurando mecanismos que se adequassem à necessidade atual de disponibilidade de serviço de forma a interferir o mínimo

possível no desempenho de servidores, foi estudada e implementada uma solução que aumenta a disponibilidade se servidores através de interrupção e reinício de requisições.

Como caso de estudo para aplicação dos experimentos foi aplicada a técnica de interrupção e reinício no WS-DSAC, uma plataforma desenvolvida por Serra [Serra 2005 a] em clusters de servidores Web. A plataforma WS-DSAC possui um mecanismo de balanceamento de cargas em dois níveis que favorece uma diferenciação de serviços objetivando a utilização eficiente dos recursos de processamento disponíveis.

A previsão da carga de trabalho é uma das fases do planejamento de capacidade, portanto, os objetivos da fase de previsão são determinados implicitamente pelos objetivos do projeto de planejamento da capacidade. Vários critérios podem ser usados para a seleção das técnicas de previsão. O primeiro refere-se à extensão da previsão (por exemplo, curto, médio e longo prazos). Outros critérios dizem respeito à disponibilidade de dados históricos, ao padrão dos dados e à precisão desejada.

O WS-DSAC serve-se de um cálculo de estimação de cargas denominado coeficiente de reatividade. Esta métrica favorece o mecanismo de balanceamento de carga na distribuição das requisições de acordo com a classe do serviço processada.

Utilizada no monitoramento da temperatura em reatores de água pressurizada [Antonopoulos 1999], o coeficiente de reatividade trata-se de uma técnica simples, mas eficiente para a medição de carga de servidores.

A escolha do coeficiente de reatividade deve-se ao fato do tempo de resposta dos pedidos não ser um bom instrumento de medida quando conteúdos dinâmicos são distribuídos igualmente sobre o servidor. O tempo de resposta depende do tipo de serviço e carga do servidor num determinado momento. Cada pedido pode ter diferentes tempos de resposta e o tempo de resposta de um pedido

de um serviço específico varia de acordo com a carga do servidor quando do tratamento. O "coeficiente de reatividade" fornece uma idéia da tendência de carga do servidor, estimando o tempo de espera de uma tarefa pela CPU.

O mecanismo de balanceamento da carga usa a informação fornecida através de informações a cerca da carga do servidor. A carga do nó é estabelecida pelo valor do coeficiente de reatividade que fornece as tendências da carga de cada computador da plataforma. Este coeficiente fundamenta-se no tempo médio de espera da CPU que está relacionado com a carga do servidor. Para eliminar os efeitos instantâneos de variações da carga da máquina, repete-se esta operação diversas vezes.

A partir de observações relatadas por Olejnik [Olejnik 2002], comprovou-se que os mecanismos que utilizaram essa medida obtiveram uma melhora no balanceamento da carga em aplicações distribuídas.

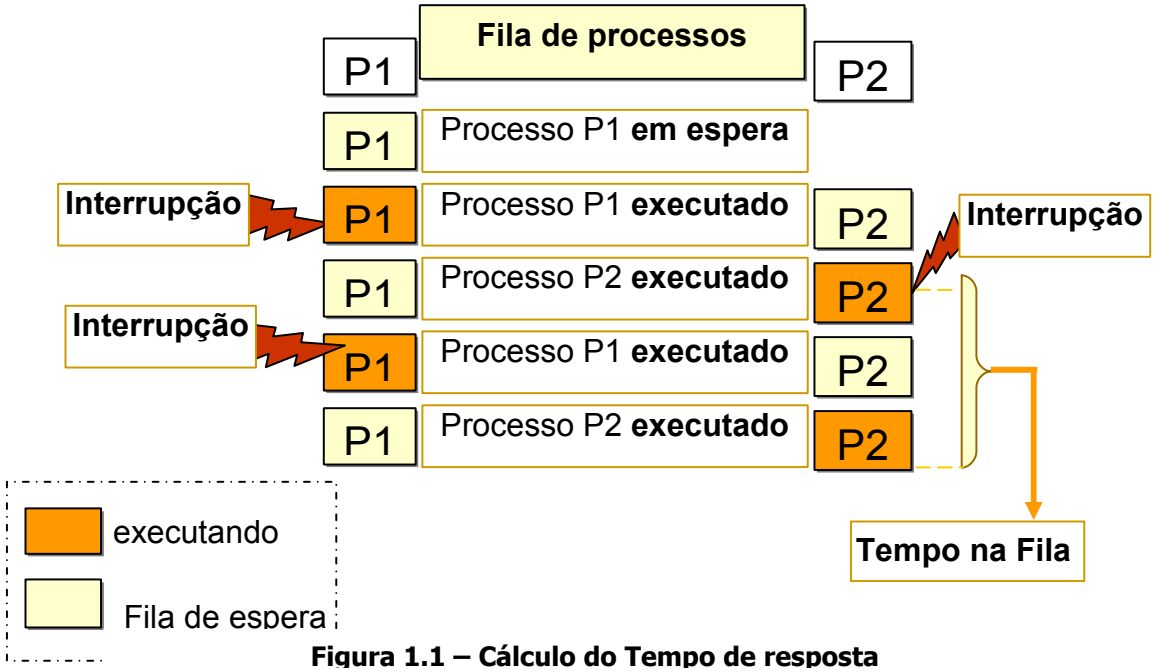


Figura 1.1 – Cálculo do Tempo de resposta

Para explicar como se calcula o tempo de resposta pelo coeficiente de reatividade considere dois processos P1 e P2 (Figura 1.1).

O processo P1 quando é submetido para execução é inserido em uma fila de processos aptos para executar, está pronto para rodar. Quando o mesmo processo é selecionado passa do estado pronto para o estado rodando. Quando faz uma chamada de sistema P1 perde o processador (por exemplo, solicitou uma operação de entrada e saída) e passa para o estado bloqueado. Durante o tempo em que o sistema operacional trata a chamada de sistema um novo processo (P2) é selecionado para execução. Ao término da chamada de sistema P1 passa para o estado pronto para rodar e é inserido na fila de aptos. O tempo em que um mesmo processo é interrompido e passa a ser reativado é o tempo de resposta na fila de processos. Uma média desse tempo de resposta é calculado, Desta forma pode-se ter noção do tamanho da fila de processos e a capacidade de processamento dos mesmos.

Parte do algoritmo do cálculo do coeficiente de reatividade é demonstrado na Figura 1.2. Nela é possível notar o início da marcação de um processo e seu fim e atualização da carga máxima para o cálculo da média do tempo de resposta.

```
enquanto (verdadeiro) {
tempo_inicio;
suspende();
tempo_fim;
carga = tempo_fim - tempo_inicio;
      se (carga > 0) {
            conta();
            cargaAcumulada(carga);
          }
          se (carga > cargaMax()) {
                      cargaMax(load);
```

Figura 1.2 – Algoritmo do coeficiente de reatividade

Para contextualizar a implementação do ambiente, são referidos no próximo tópico, detalhes desta plataforma.

1.2. Contexto do Trabalho

O mecanismo de interrupção e reinicio serve como um suporte alternativo de alta disponibilidade, assegurando que o sistema esteja funcionando e pronto para uso em um dado instante de tempo [Conectiva].

A técnica de interrupção e reinício foi empregada para possibilitar a implementação de um bloco funcional de alta disponibilidade. Procurou-se através dela explorar os benefícios dos reinícios como forma de disponibilizar serviços mesmo em momentos de carga elevada.

Quando ocorre sobrecarga no servidor, algumas requisições têm suas respostas por algum tempo sustadas vindo posteriormente

a serem atendidas quando o serviço interrompido novamente é disponibilizado. A disponibilização dos serviços suspensos é feita quando a carga do servidor atinge um nível de carga denominado compartilhado. O nível compartilhado significa que os recursos estão disponíveis para processamento de cargas maiores e atendimento de requisições de outras classes de serviços Assim, pode-se controlar o nível de saturação do servidor e a disponibilidade do serviço em servidores.

Os reinícios possuem três vantagens importantes. Primeiramente, ele faz com que um processo retorne ao seu estado inicial, ou se torne imune a problemas. Em segundo lugar, os reinícios possuem a propriedade de elevar a confiabilidade do sistema. Na maior parte dos servidores Apache da internet, a técnica de suspensão e reinícios é utilizada a fim de controlar o consumo dos recursos sob uma carga pesada de processamento. A outra vantagem é que reiniciar constitui-se em um mecanismo simples, de fácil implementação e de fácil recuperação.

O mecanismo de interrupção e reinício é empregado em tempo real possibilitando reativações de requisições de forma rápida e eficiente. Segundo Candea [Candea 2001], essa filosofia pode reduzir em até 5 (cinco) vezes o tempo de reparo, dependendo do sistema. O mecanismo de interrupção e reinício requer pouco esforço computacional e favorece uma alta disponibilidade de serviços.

Para atingir os objetivos propostos com o emprego do mecanismo de interrupção e reinício, utilizou-se uma plataforma distribuída formada por clusters denominada WS-DSAC. Esta plataforma oferece diferentes níveis de QoS baseando-se na diferenciação de serviços apresentada em [Serra 2004a]. O mecanismo de interrupção e reinicio foi inserido na plataforma WS-DSAC, onde seu princípio fundamenta-se na suspensão de requisições de serviços quando o servidor alcança elevadas cargas. Este mecanismo permite garantir que todas as solicitações sejam

atendidas reativando as requisições anteriormente suspendidas. Maiores detalhes sobre o composição e funcionalidades desta plataforma encontra-se no Capítulo 3 desta dissertação.

1.3 Objetivos

Este trabalho visa a implantação de um mecanismo capaz de controlar os níveis de QoS e permitir que serviços sejam providos de forma ininterrupta.

O projeto visou incrementar a uma plataforma voltada para a Web e construída sob clusters de computadores, características que propiciem uma alta disponibilidade e qualidade no serviço.

A fim de atender aos objetivos finais apresentados acima, foram estabelecidas metas como objetivos intermediários conforme mostrados a seguir:

- Usar mecanismo de interrupção e reinício, como ferramenta no tratamento de sobrecarga de servidores;
- Adicionar características que possibilitem assegurar a disponibilidade de serviços em clusters;
- Utilizar o "Coeficiente de Reatividade" como parâmetro de medição para analise;
- Estudar a viabilidade da aplicação do mecanismo de interrupção e reinício em servidores Web.

1.4 Organização da dissertação

Este trabalho está estruturado em 6 (seis) capítulos, esquematizados por tópicos relativos a conceitos teóricos, implementação, experimentação e exposição de resultados.

No Capítulo 1 situou-se o problema que motivou a pesquisa, traçando as metas a serem alcançadas ao término do presente trabalho.

No Capítulo 2 são apresentados conceitos básicos sobre qualidade de serviços e clusters.

No Capítulo 3 fez-se a contextualização do ambiente de trabalho com a exposição da plataforma WS-DSAC e a explanação da comunicação entres os seus módulos.

No Capítulo 4 são apresentadas as técnicas empregadas na solução de problemas de disponibilidade e as que serão utilizadas no cumprimento dos objetivos deste trabalho. Também é descrita a nova arquitetura da plataforma WS-DSAC com o incremento do novo módulo encarregado pelo controle de QoS e alta disponibilidade.

No Capítulo 5 são apresentados os experimentos e resultados obtidos e analisados durante as situações críticas com aplicação da técnica recursive restartability.

No Capítulo 6 são apresentados as conclusões e trabalhos futuros decorrentes desta pesquisa.

2. Clusters

2.1 Introdução

Nos últimos anos, as pesquisas dos sistemas de computadores estiveram focalizadas no ganho de desempenho, e isto rendeu uma melhoria significativa na capacidade de processamento [Hennessy 2002]. Como tecnologia alternativa, os clusters constituem-se de sistemas computacionais de alto desempenho formados por computadores interligados por uma rede local [Senger 2004] [Mello 2003].

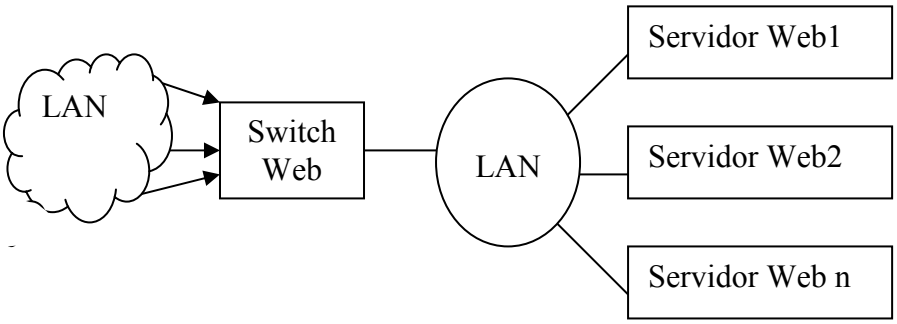


Figura 2.1 – Visão Geral do cluster WS-DAC

2.2 Razões para se utilizar Clusters

Considerando a inviabilidade de processamento de tarefas que exigem um grande esforço computacional em uma arquitetura de computadores normal, podemos notar que entre as razões da utilização de clusters se deve a sua maior flexibilidade e escalabilidade, a sua tolerância a falhas devido a mecanismos que visam contornar situações de parada e, finalmente uma arquitetura que permite fácil manutenção além de serem de custo ínfimo quando comparadas aos supercomputadores, podendo alcançar desempenho próximo destes devido ao balanceamento de carga. Com nodos independentes, o cluster permite uma fácil manutenção, desta forma

o acréscimo e a remoção de nodos, drivers ou equipamentos tornam-se muitos menos dispendiosos.

2.3 Aplicações de Clusters

Basicamente, clusters são aplicados em qualquer sistema que exija processamento pesado. Um exemplo é o cluster implantado no departamento de química da Universidade Stanford, nos EUA, que organiza o Projeto Folding (folding.stanford.edu), que estuda simulações da dinâmica molecular de proteínas.

Outra aplicação no campo de pesquisa é desenvolvida pela United Devices [Projects] visando uma cura para o câncer. Cluster de computadores pode ser aplicado em outros campos como comércio, indústria e serviços devido sua eficiente distribuição de carga e baixo custo relativo.

Geralmente aplicações críticas, em que os serviços têm que estar disponíveis e/ou processados o mais rápido possível, utilizam as tecnologias de cluster, desde que devidamente configurados para situações de falhas graves.

Na indústria cinematográfica, em gráficos de altíssima qualidade e animações, os clusters paralelos são de fundamental importância.

Um exemplo foi o filme "Titanic" de 1997, que foi renderizado utilizando essa tecnologia nos laboratórios da Digital Domain. [Linux Magazine 1999][ASM 1999][Journal].

2.4 Funcionamento e Arquitetura

Um sistema de clusters pode ser dedicado para a solução de um só processamento ou como uma máquina paralela compartilhada entre vários usuários. O cluster de computadores funciona como um sistema computacional único embora de processamento paralelo.

Em geral, em clusters utilizam-se hardawares idênticos. Porém uma outra alternativa seria um nó principal mais rápido em relação aos demais [Linux 2004].

Hardwares idênticos simplificam a instalação e configuração dos clusters e seus conjuntos. Essa simetria do sistema em cada nodo facilita a gerência dos clusters tornando menor a complexidade da codificação por não ser preciso tratar nodos de configurações de hardware e software distintas.

Na a arquitetura do cluster utilizado em nosso trabalho apenas o nodo principal realiza o monitoramento do dos demais nodos.

2.5 Tipos de Clusters

Os clusters são classificados basicamente em 4 (quatro) categorias de acordo com sua finalidade [Pitanga 2002]:

- Alta Disponibilidade;
- Balanceamento de carga;
- Combinação alta disponibilidade e balanceamento de carga;
- Processamento Distribuído ou Processamento Paralelo.

O modelo de Clusters de alta disponibilidade é estruturado para fornecer uma disponibilidade de serviços e recursos de forma ininterrupta através do uso da redundância implícita ao sistema. De modo geral, se um nó do cluster vier a falhar, suas aplicações ou serviços podem ficar disponíveis para serem executados em outro nó. Geralmente esse modelo é usado para base de dados de missões críticas, servidores de arquivos e aplicações.

O modelo de clusters de balanceamento de carga distribui as requisições de recursos entre as máquinas que compõem o cluster.

Estando todos os nodos aptos a atender os pedidos, no caso de falha em um nó, as requisições são redistribuídas entre os nós de menor carga no momento.

O modelo de combinação da alta disponibilidade e balanceamento de carga trata-se de uma combinação das características dos modelos de alta disponibilidade e de balanceamento de carga. Essa combinação aumenta a disponibilidade e escalabilidade de serviços e recursos.

O modelo de cluster de processamento distribuído ou processamento paralelo é aplicado em ambiente de grandes tarefas computacionais apresentando grande desempenho através da distribuição das tarefas entre os nodos, apresentando um alto poder de processamento.

2.6 Conclusão

Otimizar o desempenho de redes operacionais envolve objetivos orientados ao tráfego e a recursos [Awduche 1999]. A tecnologia de clusters propicia uma solução prática e aplicável em servidores de diversos fins. Ela apresenta diversas vantagens, como incremento da sua escalabilidade, melhor gerenciamento de recursos, fácil manutenção, possibilitando ainda características de alta disponibilidade viabilizando sua aplicação em serviços de grande criticidade.

3. Plataforma WS-DSAC

3.1 Introdução

Este capítulo será dedicado a apresentação do WS-DSAC, uma plataforma desenvolvido por [Serra 2005a][Serra 2005b] em Clusters de servidores Web que tem entre outros atributos a capacidade de balancear a carga imposta ao sistema, e que faz uma diferenciação da QoS entre os clientes. O mecanismo WS-DSAC constitui-se numa solução que contribui para o aumento da QoS de aplicações disponibilizadas na Internet bem como para a diferenciação de usuários de acordo com a necessidade e a capacidade de pagamento de cada um.

3.2 Contexto do Trabalho

Como caso de estudo para aplicação dos experimentos desta dissertação, utilizou-se a plataforma WS-DSAC com o objetivo de se implantar nesta plataforma mecanismos de alta disponibilidade de serviços e garantia de QoS. A Implementação do WS-DSAC foi feita com a tecnologia Java 2 SDK, Standard Edition 5.0 [Java], Java Remote Method Invocation (Java RMI) na invocação de métodos[RMI], Apache Tomcat 5.5.9 [Tomcat].

3.3 Visão Geral da Plataforma

A plataforma WS-DSAC (Figura 3.1) é composta por um conjunto de elementos básicos: “Class Switch”, “Cluster Gateways” e “Servidores Web”.

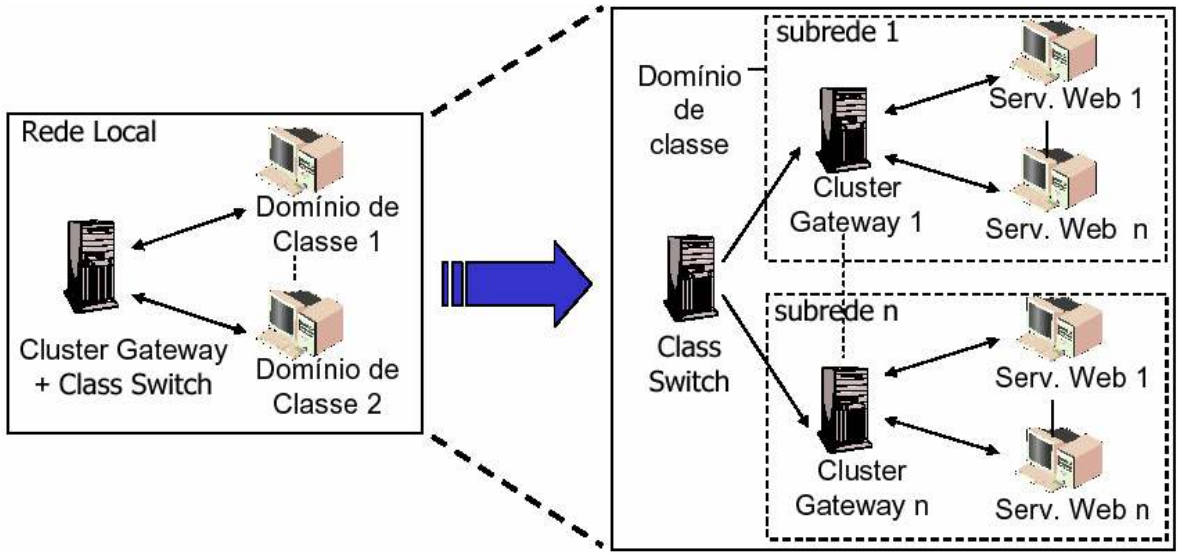


Figura 3.1 – Visão Geral da Plataforma.

O "Class Switch" assume o papel de classificar as requisições e controle de admissão destas. As requisições acontecem sob o protocolo HTTP, e são categorizadas por classes de serviços e enviadas a um "Cluster Gateway" especificado que é responsável por balancear a carga entre os servidores.

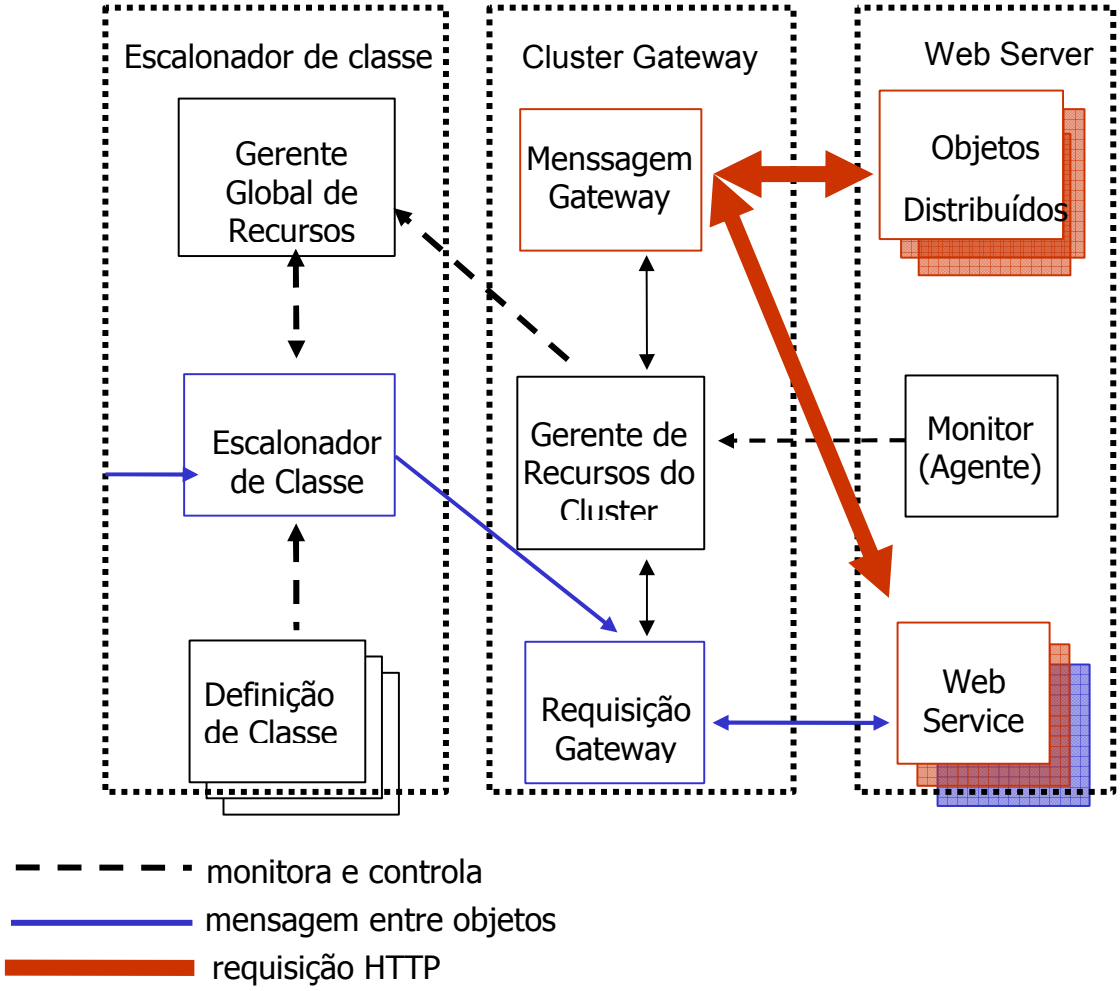


Figura 3.2 – Comunicação entre componentes da Plataforma.

A cada classe de serviços está associado um limite de carga, desta forma os serviços têm níveis de QoS diferenciados. Estes serviços estão distribuídos entre os servidores Web podendo dispor-se de serviços Web e objetos distribuídos.

3.4 Componentes da Plataforma

A plataforma WS-DSAC divide-se basicamente em três mecanismos:

- Mecanismos de Monitoramento;
- Mecanismos de Controle de Admissão e Balanceamento de Cargas;
- Serviços Administrativos.

3.4.1 Mecanismos de monitoração e gestão de QoS do WS-DSAC

O mecanismo de monitoramento é formado por três elementos funcionais: o GGR (Gerenciador Global de Recursos), o GRC (Gerenciador de Recursos do Cluster) e o MA (Monitor Agente) (Figura 3.2).

O GGR solicita e mantém a informação de carga de cada GRC, ou seja, de cada cluster.

O GRC tem a missão de monitorar e estimar a carga do cluster de uma classe. Cada domínio de classe ("Class Cluster Domain") executa um GRC (Gerenciador de Recursos do Cluster) que solicita informações de carga de cada servidor Web periodicamente. Ele analisa todos os "coeficientes de reatividade" dos servidores do cluster e faz uma estimativa para a carga do cluster inteiro.

O MA tem por função monitorar e estimar a carga de cada servidor Web tendo a responsabilidade de registrar o servidor Web em um cluster de classe de serviços específico, e solicitar a sua inclusão ao GRC no domínio da mesma classe. Ao MA, cabe a tarefa de monitorar a carga do servidor Web com base na técnica de monitoramento denominada "Coeficiente de Reatividade" [Olejnik 2002]. Mais detalhes sobre essa técnica serão posteriormente

apresentados. Por fim, o MA ainda envia informações acerca da carga de cada servidor Web dentro de intervalos fixos de tempo.

3.4.2 Mecanismo de Controle de Admissão e Balanceamento de Cargas

O mecanismo de controle de admissão e balanceamento de cargas está disposto em três partes: MOG (Mensagem do Objeto Gateway), HRG (HTTP Requisição Gateway) e EC (Escalonador da Classe) (Figura 3.2).

As requisições que são advindas do protocolo HTTP são recebidas pelo EC e são separadas por categorias passando por um controle de admissão. As requisições do EC são direcionadas pelo HRG ao servidor de menor carga no momento.

O MOG atua re-direcionando os objetos do um cluster com pretensão de balancear a carga entre os nodos servidores.

As requisições oriundas do EC (Escalonador da Classe) são distribuídas pelo HRG (HTTP Requisição Gateway) entre os servidores Web de acordo com a informação de carga obtida pelo GRC.

Estas requisições vindas do EC são provenientes de clientes e são classificadas conforme a classe de serviço a que pertence. Feito isso, o EC consulta a carga pelo GGR. Sendo possível garantir um serviço sem prejudicar a sua qualidade, o EC aceita o processamento da informação, caso contrário notifica ao cliente a impossibilidade de processamento daquela requisição.

3.4.3 Serviços Administrativos

Os serviços de administração da plataforma são agrupados em: Serviços de Instalação, Serviços de Definição de Classes e Serviços de Monitoramento dos Clusters. Os serviços de instalação do WS-DSAC facilitam a instalação de serviços Web e objetos distribuídos na

plataforma. Estes serviços possibilitam a criação dos mecanismos de balanceamento de carga. As classes e as características destas classes de serviços são definidas pelo administrador que associa a cada uma delas um determinado coeficiente de reatividade. O serviço de administração ainda apresenta as tarefas de monitorar a carga e ajustar a política de distribuição de recursos baseado em dados estatísticos tais como a taxa de requisições recusadas, média do coeficiente de reatividade por classe de serviços do sistema.

3.5 Algoritmo WS-DSAC

Formalmente o mecanismo WS-DSAC pode ser definido através dos seguintes parâmetros:

T1, T2, T3, Tn,… - Representam intervalos de tempo no decurso da admissão ou rejeição de solicitações.

N_c – Representa o número de classes de serviços diferentes (class clusters), em que necessariamente ($N_c > 1$).

N_{si} – Quantidade de servidores pertencentes ao domínio da classe "i", em que ($N_{si} > 0$).

R_i^{ac} – O valor máximo permitido ao "coeficiente de reatividade" da classe "i", em que ($1 < i <= N_c$). A partir deste valor ($i = N_c$)o serviço passa a trabalhar em « modo saturado ».

R_{ki}^{em} – Este parâmetro informa o valor do "coeficiente de reatividade", calculado no período k. A partir deste valor o cluster passa a trabalhar em « modo exclusivo ». Com o coeficiente abaixo deste valor o cluster trabalha em modo compartilhado.

r_{kij} – Esta variável descreve a carga média do "coeficiente de reatividade" do servidor Web "j" dentro do cluster da classe "i", ao longo de um período k em que ($1 < i <= N_c$) e ($1 <= j <= N_{si}$).

r_{ki} – Simboliza a média da carga do coeficiente de reatividade dos servidores pertencentes a um domínio da classe "i" calculada ao logo de k períodos. Ou seja:

$$r_{ki} = \left(\frac{\sum_{j=1}^{Nsi} r_{kij}}{N_{si}} \right) \qquad (1)$$

ρ_{kij} – Descreve a carga prevista para o período de tempo (k+1) do servidor Web "j", pertencente ao domínio da classe "i", calculada durante um tempo k por meio de uma função de

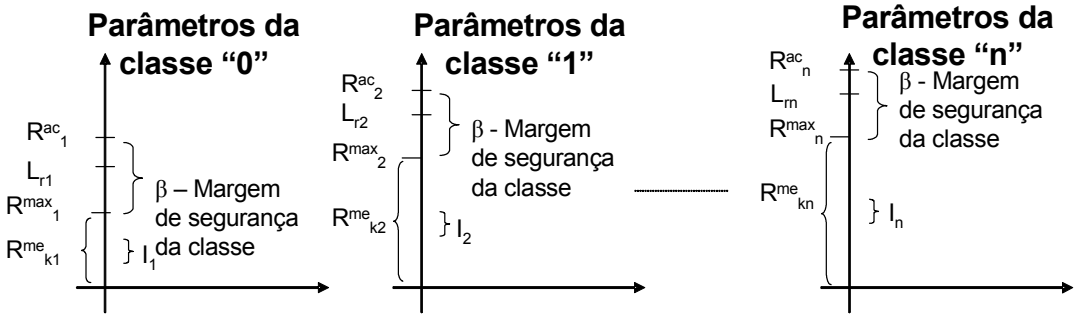


Figura 3.3: Parâmetros utilizados para o cálculo do "modo de trabalho".

controle de admissão f_{ac} após o intervalo k e antes que o intervalo (k+1) inicie, ou seja $\rho_{kij} = f_{ac}(k+1)$.

A função de controle de admissão $f_{ac}(k+1)$ e é dada por :

$f_{ac}(1) = R_i^{ac}$.

$f_{ac}(k+1) = (1-\alpha) * f_{ac}(k) + \alpha * r_{kij}$.

O parâmetro "α" é o coeficiente de ponderação podendo assumir valores entre 0 e 1. Este parâmetro calcula o índice de estabilidade do mecanismo [Carpenter 2002].

ρ_{ki} – A média da carga prevista é assim formulada:

$$\rho_{ki} = \left(\frac{\sum_{j=1}^{Nsi} \rho_{kij}}{N_{si}} \right)$$

M_{ki} – O modo de trabalho do cluster (Figura 3.3) da classe "i" durante o período de tempo (k+1) calculado a partir da $f_{ac\text{-}mode}$ (função de cálculo do modo de trabalho), dada por :

$$f_{ac\text{-}mode}(k) \begin{cases} 0 \text{ (se } \rho_{ki} <= R_{ki}^{em} \text{) ("compartilhado")} \\ = 1 \text{ (se } R_{ki}^{em} < \rho_{ki} <= R_i^{ac} \text{) ("exclusivo")} \\ 2 \text{ (se } \rho_{ki} > R_i^{ac} \text{) ("saturado")} \end{cases}$$

$\rho_{k\ min}$ – É responsável por repassar a carga estimada do "class cluster" menos sobrecarregado em um determinado intervalo de tempo.

O mecanismo funciona da seguinte forma: a cada k-ésimo período de tempo o "class cluster gateway" interroga todos os servidores do seu domínio para obter as cargas atuais (r_{kij}) e estimar as cargas do próximo período (ρ_{kij}). Em seguida, cada "class cluster" calcula seu "ρ_{ki}". Baseado nestes cálculos o modo de trabalho de cada "class cluster" durante o próximo período de tempo (k+1) é determinado.

Quando uma requisição chega durante o k-ésimo intervalo de tempo, o mecanismo WS-DSAC é aplicado. Primeiro o "class switch" identifica qual é a classe da requisição. Dado que a requisição pertence à classe "i" e que o cluster "m" é o menos carregado, o "class switch" executa o algoritmo a seguir:

Se ($\rho_{k\ min}$ <= R_i^{ac}) e (M_{km}= 0) então
  Enviar a requisição para o cluster gateway "m";
Senão
  Se (ρ_{ki} <= R_i^{ac}) então
    Enviar a requisição para o cluster gateway da classe "i";
  Senão
    Enviar mensagem "Requisição rejeitada" para o cliente;
  Fim se
Fim se

Quando o "cluster gateway" eleito pelo algoritmo acima recebe a requisição redirecionada pelo "class switch" ele envia a mesma para o servidor menos carregado do cluster, ou seja, para o servidor que possui o menor "ρ_{kij}".

3.6 A nova arquitetura e sua descrição

Neste tópico, é apresentada a evolução da arquitetura do WS-DSAC, uma plataforma cluster desenvolvida por Serra [Serra 2005a] [Serra 2004b]. Nesta nova arquitetura (WS-DSAC 2.0) foi agregado

um novo elemento denominado “Gerente de carga” (ou simplesmente Gerente). Este elemento assume o papel de um escalonador de requisições mediante seu respectivos os processos e se propõe a otimizar a capacidade do servidor e a disponibilidade de serviços.

A necessidade do acréscimo do novo elemento (Gerente de carga) decorre do fato de se evitar a sobrecarga nos nodos servidores.

Experimentos com o WS-DSAC 1.0 mostraram que o mecanismo é capaz de distribuir de forma eqüitativa os recursos em momentos de baixa utilização e de assegurar os contratos de QoS estabelecidos com cada classe de serviços em momentos de sobrecarga.

Porém o WS-DSAC 1.0 apenas procura administrar a sobrecarga dos nodos servidores, não possuindo uma política de tratamento neste caso. Desta forma o aumento da QoS de aplicações disponibilizadas na Internet pelo WS-DSAC 1.0 deriva de sua política que enfoca a diferenciação de usuários mas não evita a sobrecarga em situações de intensas solicitações de serviços.

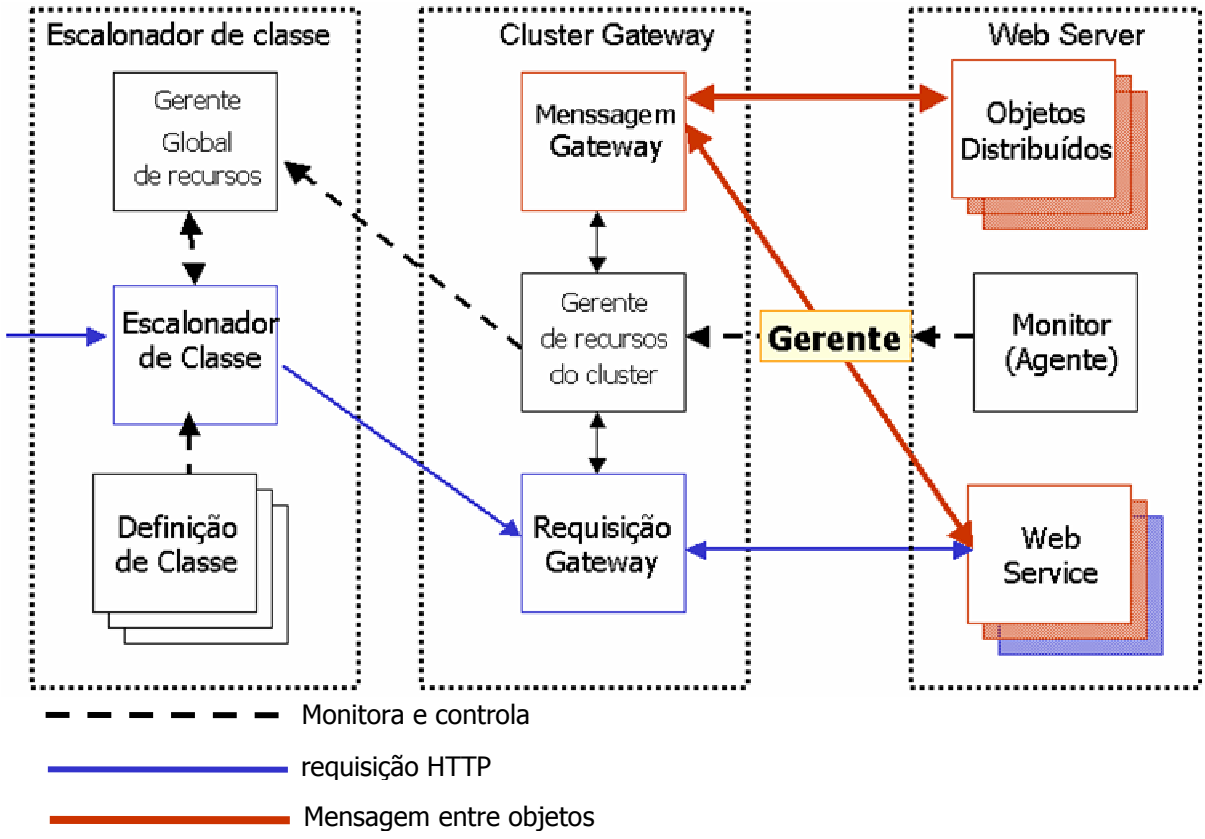


Figura 3.4-Nova arquitetura da plataforma WS-DSAC

Por esse motivo o Gerente de carga (Figura 3.4) tem entre suas principais funcionalidades o escalonamento (interrupção e reinício) das requisições a partir do coeficiente de reatividade, parâmetro de QoS adotado no WS-DSAC [Serra 2004a][Serra 2004b][Serra 2004c].

3.6.1 Gerente de carga com interrupção e reinício de processos

A técnica de interrupção e reinício de processos foi empregada para possibilitar a implementação de um bloco funcional de alta disponibilidade. Procurou-se através dela explorar os benefícios dos reinícios como forma de controle na qualidade nos serviços oferecidos pelo servidor. Sua implementação foi inspirada na técnica recursive restartability, aplicada em tolerância a falhas [Candea 2001][Candea 2002], mas que ainda não foi empregada em servidores Web.

Apesar de terem os mesmos princípios, a técnica de interrupção e reinício de processos distingue-se da recursive restartability pelo fato da primeira forçar a interrupção do processo para em seguida reativá-lo em busca de melhor desempenho no servidor, enquanto que a segunda acontece em caso de falhas sistêmicas (Figura 3.5).

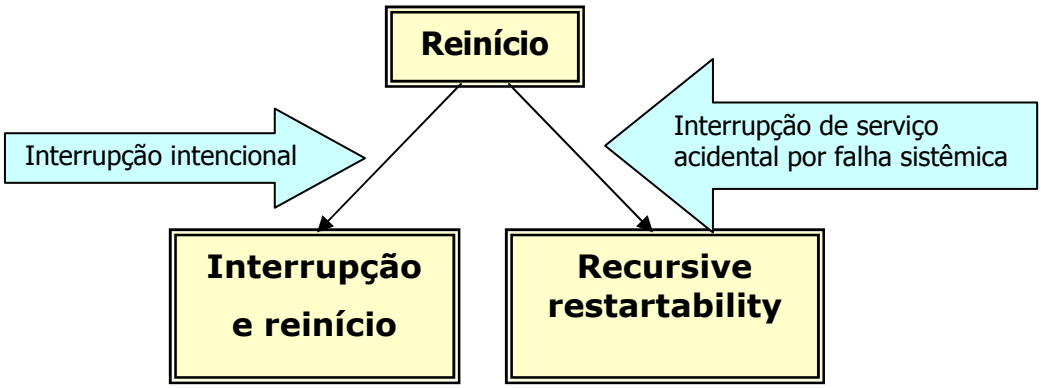


Figura 3.5-Abordagens com reinícios

No tratamento de "condições de corrida" (livelock), por exemplo, a melhor e mais freqüente maneira de solucionar estes problemas por parte dos administradores de sistemas é através de reínicios [Brewer 2001] [Reeves 1998].

Um exemplo prático disso ocorreu em julho 1998, quando o USS Yorktown, um navio de guerra dos Estados Unidos que estava na guerra do Golfo perdeu o controle de seu sistema da propulsão devido a uma cadeia de eventos iniciados com a sobrecarga de dados

em seus computadores [Candea 2004a][Candea 2004b]. A aplicação da técnica recursive restartability poderia ter restaurado o sistema, evitando a necessidade do navio de ser rebocado para o porto [DiGiorgio 1998][Candea 2003].

Quando se realiza a técnica de reinícios dentro de um tempo reparo razoável ela torna-se uma solução eficientemente rápida na recuperação de estados defeituosos.

Segundo Candea [Candea 2001], essa filosofia pode reduzir em até 5 (cinco) vezes o tempo de reparo, dependendo do sistema.

O reinício de processos é uma técnica de software que tem a capacidade de recuperar o estado inicial de um processo. Uma das maiores vantagens desta técnica é que em casos de falhas temporais ou em qualidade de serviços ineficientes, os reinícios são limitados apenas à determinados processos, ficando os demais processos isolados das interrupções. Este trabalho é pioneiro no tratamento de sobrecargas em servidores Web com reinícios de processos tendo como base o coeficiente de reatividade de cada nodo em cluster de computadores.

A aplicação da interrupção e reinicialização dos processos objetiva evitar que algum nodo do cluster do WS-DSAC atinja o modo saturado (sobrecarga do servidor). O êxito desse objetivo significa o atendimento de todas as requisições solicitadas, bem como o aumento da disponibilidade de serviços (processos) prioritários e sua conseqüente melhora na qualidade de serviço. O algoritmo empregado na implementação do Gerente de carga será explanado no Capítulo 4 desta dissertação

3.7 Conclusão

A plataforma WS-DSAC possui um mecanismo um balanceamento de cargas em dois níveis que favorece uma diferenciação de serviços objetivando a utilização eficiente dos recursos de processamento disponíveis O WS-DSAC serve-se de um

cálculo de estimação de cargas denominado coeficiente de reatividade. Esta métrica de carga favorece o mecanismo de balanceamento de carga na distribuição das requisições de acordo com a classe do serviço processada. O WS-DSAC porém sob condições de sobrecarga pode chegar a um estado denominado saturado. Isto faz com que requisições não sejam.atendidas A aplicação da interrupção e reinício de requisições permite ao sistema recuperar-se de estados defeituosos constituindo-se em uma solução que requer pouco esforço computacional e favorece uma alta disponibilidade de serviços.

4. Implementação de um mecanismo de alta disponibilidade no ambiente Java-Linux

4.1 Introdução

Em computação de alto desempenho, o processamento de uma aplicação multimídia é normalmente efetuado por meio de clusters de computadores.

Em clusters, o processamento paralelo permite que vários processos de uma mesma aplicação sejam executados simultaneamente em diferentes máquinas. Nesta dissertação é apresentado um estudo e a implementação do monitoramento de um cluster. Foram utilizadas duas máquinas servidoras monitoradas por uma terceira máquina.

A monitoração de serviços é feita através do Monitor, um escalonador de testes que pode acompanhar o comportamento de máquinas e serviços de forma rápida e ágil, enviando a sobrecarga da máquina e permitindo níveis aceitáveis de qualidade dos serviços. Em um comportamento considerado alarmante, o Monitor interferirá evitando uma situação crítica automaticamente, ou reiniciará uma máquina, caso a falha ocorra de forma a ser impossível seu tratamento. Essa monitoração centralizada é geralmente usada quando se tem um número relativamente pequeno de servidores [PECW].

Alguns dos índices que têm sido usados, estudados, analisados e avaliados:

- a média do tamanho da fila de CPU em um intervalo de tempo;
- a utilização da CPU;
- a quantidade de memória disponível;
- o tempo de resposta.

4.2 Controle dos Processos

4.2.1 Requisições controladas por Threads

Cada requisição gera um ou mais processos. Na implementação, as requisições do usuário são controladas por threads, processos leves que compartilham um mesmo espaço de endereçamento [Threads], podendo funcionar periodicamente ou através de eventos esporádicos. A implementação empregada baseou-se em um modelo de "multithread" que cria uma nova linha de execução para invocar as requisições solicitadas pelo usuário.

Para reproduzir um cenário de sobrecarga por meio de requisições de usuários enviadas ao servidor Web, foi implementado um gerador de tráfego (Figura 4.1) no intuito de estudar o comportamento da carga do servidor mediante a aplicação da técnica de interrupção e reinício.

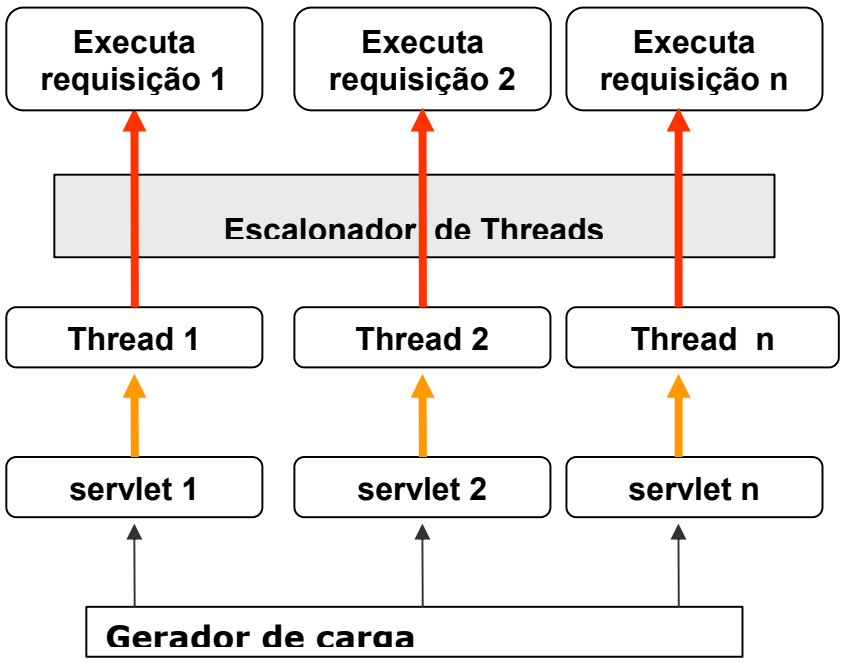


Figura 4.1-Controle sobre os processos com threads no WS-DSAC

A execução e interrupção das requisições geradas pelo simulador são controladas por threads, processos leves que

compartilham um mesmo espaço de endereçamento [Threads]. Por ocasião da implementação trabalhou-se com várias linhas de execução (multithread) que operam de maneira síncrona na ativação e desativação de processos.

A principal tarefa desses threads é escalonar processos (requisições), acionando-os após interrupção dos mesmos. A interrupção ocorre quando o servidor encontra-se sobrecarregado.

Enquanto o servidor Web recebe as requisições, as mesmas serão atendidas de acordo com sua prioridade. Os threads podem assumir quatro estados possíveis: novo, executando, bloqueado ou morto. O primeiro estado refere-se às requisições solicitadas pelo cliente, através de requisições. Caso o servidor não esteja sobrecarregado, a solicitação é atendida sendo executada, correspondendo ao segundo estado possível de um thread (Figura 4.2).

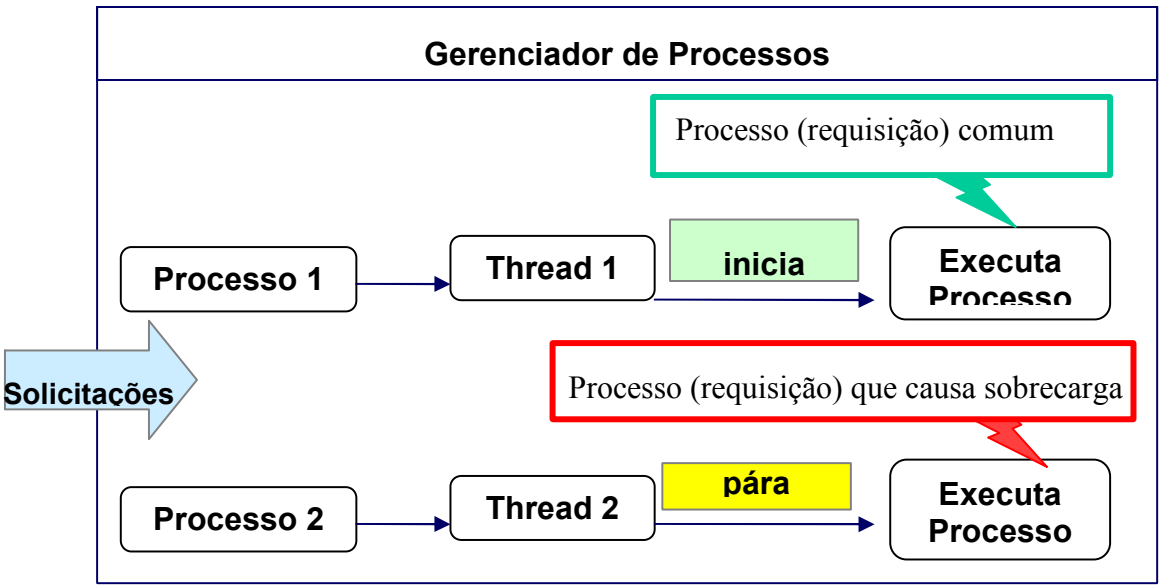


Figura 4.2-Interrupções de processos com threads no gerente de carga

Quando há sobrecarga no servidor um determinado processo é suspenso mediante comando via thread ("yield") e logo em seguida reiniciado. Neste exemplo (Figura 4.2) as duas cargas são executadas

e controladas por threads em paralelo. Apenas uma das cargas sofre reinícios enquanto que a outra processa ininterruptamente durante todo o tempo da simulação.

4.3 Algoritmo do gerente de carga (Mecanismo de Interrupção e Reinício)

O algoritmo do gerente de carga (Figura 4.3) possui um mecanismo de controle de acordo com a carga do servidor. Esse controle que se constitui de suspensão e reativação de um processo mediante thread é efetuado quando se atinge uma carga no servidor com coeficiente de reatividade maior que 300 ms (R_i^{ac}, seção 3.5). O motivo é pelo fato do coeficiente de reatividade 300 ms ser o valor máximo permitido. Este algoritmo apresenta o comportamento da carga mediante a técnica de interrupção e reinício.

<table>
<tr><th colspan="2">Carga < 300 ms</th><th>300 ms < Carga < 600 ms</th><th>Carga > 600 ms</th></tr>
<tr><th>C = 0</th><th>C > 0</th><td rowspan="2">Ativa P1;
Para P2;
C = C + 1</td><td rowspan="2">Ativa P1;
P2 em Espera;
Para P2;
C = C + 1</td></tr>
<tr><td>Ativa P1;
Ativa P2;</td><td>Ativa P1;
C = C –1;
Ativa P2;</td></tr>
</table>

Figura 4.3- Tratamento dos eventos do gerente de carga

A reativação de um processo (requisição) suspenso ocorre quando a carga atinge um índice menor que 300 ms (Figura 4.3). Um contador de interrupções complementa a condição para que haja a reativação do processo que foi suspenso. Para efeito de estudos quanto a viabilidade do método bem como do seu comportamento, a requisição é identificada mediante o nome da thread associada a requisição em questão.

Na Figura 4.3 são apresentados os intervalos de carga em que o processo P2 sofrerá interrupção e reinício. Nesta figura o processo P2 será ativado neste caso específico quando a carga for menor que 300 ms. Se não tiver sofrido reinício "C" que representa a quantidade de interrupções será igual a zero. Caso tenha ocorrido alguma interrupção e a carga do servidor for menor que 300 ms, P2 será ativado. Desta forma o processo P2 será reativado apenas o mesmo número de vezes que for interrompido, ou seja, quando a carga tiver coeficiente de reatividade maior que 600 ms.

4.4 Critérios de simulação adotados

As simulações foram realizadas com variação que correspondia à diferenciação de classe de serviços (de admissão do serviço), quanto ao tempo de computação (execução) do(s) processo(s), quanto ao número de processos (requisições) e quanto ao método de tratamento da carga do servidor. Cada uma dessas considerações visa uma análise mais completa quanto ao comportamento no tratamento das cargas.

a) Quanto à diferenciação de classe de serviços

O WSD-SAC é suprido de um serviço que permite a diferenciação de usuários de acordo com a necessidade e a capacidade de pagamento de cada um.

Tabela 4.1- Classificação das classes por serviços

Tipos de classes de serviço	Prioridade	cliente
0	Menor	Cliente 1
1	Maior	Cliente 2

Este serviço é composto de duas classes, aqui denominadas "classe 0" (zero) e outra "classe 1" (um) que agrupam serviços para clientes distintos a fim de garantir a cada classe sua respectiva QoS (qualidade de serviço). Conforme a Tabela 4.1, a "classe 1" possui uma maior prioridade que a "classe 0". Essa diferenciação deve-se à variação dos parâmetros de perdas, retardo e variação do retardo [Serra, 2004] [Serra, 2004c].

Com essa classificação de serviços é possível efetuar um controle de admissão e diferenciação de serviços por cliente.

b) Quanto ao tempo de computação (execução) do(s) processo(s)

Algumas vezes as máquinas servidoras trabalham com certos tipos de serviços que não possuem grande prioridade. Estes serviços menos importantes por vezes trabalham em paralelo com serviços mais essenciais ou pelo menos preferenciais. Nestes casos o consumo dos recursos como a CPU e a memória é compartilhado, e em situações de carga elevada prejudicam seriamente os serviços

considerados essenciais. Nas simulações foram categorizadas também as tarefas quanto às suas prioridades, suspendendo momentaneamente, através de reinícios (suspensão e reinicializarão) [Candea, 2002], os serviços menos importantes. Na implementação associou-se os serviços mais prioritários aos processos que possuem menor tempo de computação previamente conhecido conforme a Tabela 4.2. Nela classificou-se o tipo da carga de cada processo de acordo com o seu respectivo tempo de computação. Desta maneira, um processo que leva até 100 ms foi classificado como sendo do "tipo 1". Esta classificação te duas finalidades. A primeira é separar os processos que serão interrompidos daqueles que estarão continuamente disponíveis. O segundo motivo é para facilitar a descrição dos experimentos pois estas cargas foram permutadas durante a realização dos testes.

Tabela 4.2 – Tipo de carga por tempo de processamento

Tipo da carga ponderada	Tempo de computação (ms)
1	100
2	2.375
3	2.654
4	3.211
5	20.213

c) Quanto ao número de processos (requisições)

Outro aspecto adotado para estudo do comportamento da carga foi a quantidade de processos.

Tanto na simulação com um processo quanto na simulação com dois processos as cargas foram ponderadas de acordo com a Tabela 4.3.

Na simulação com dois processos um dos processos, o de menor tempo de computação é preservando das interrupções, ficando desta forma sempre disponível, enquanto o outro processo é momentaneamente suspenso e posteriormente reiniciado conforme já descrito e mostrado na Figura 4.2. Para esse tipo de abordagem foram considerados os três estados de funcionamento do cluster:

1. Compartilhado;
2. Exclusivo;
3. Saturado.

Quando está no estado 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" atende requisições de diferentes tipos de classes. Quando o cluster passa ao "modo exclusivo", ele só aceita requisições da classe "nativa". Quando o servidor passa ao estado exclusivo significa que os níveis de cargas chegaram a um patamar onde, aceitar requisições de outras classes, pode implicar na rejeição de sessões da classe nativa. Neste caso, se as requisições de outras classes forem aceitas, recursos que são prioritariamente reservados para a classe nativa estarão sendo utilizados por sessões de outros tipos de classe. Quando o cluster passa ao estado saturado ele não aceita nenhuma

nova requisição. Isto significa que os recursos existentes não serão suficientes para garantir a

QoS assegurada às requisições em processamento, caso o mesmo aceite novas requisições. Na Tabela 4.4 são apresentados os parâmetros limitadores de cada estado a partir de valores específicos adotados para a implementação. Observa-se na Tabela 4.4 o coeficiente de reatividade (cr) dividido em intervalos conforme a carga do servidor.

Tabela 4.3 – Estados da carga do servidor

Coeficiente de reatividade (cr) para interrupção/reinícios	**Estado da carga do servidor**
cr < 300	Compartilhado
300 <= cr <600	Exclusivo
cr > 600	Saturado

d) Quanto ao método de tratamento da carga

Como descrito anteriormente, os reinícios ocorrem em uma determinada situação (Tabela 4.4). Nos estados considerados exclusivo ou saturado os processos previamente conhecidos como sendo de maior tempo de execução são interrompidos.

Para tentar minimizar o número de interrupções estudou-se duas formas de tratamento de carga denominadas "método 1" e "método 2". Este método corresponde ao procedimento comentado e apresentado na Figura 4.3. Aplicou-se além da interrupção retardar a

execução da nova solicitação de um processo de maior tempo de execução (P2, Figura 4.3).

No "método 1" não aplicou-se o atraso no tratamento da carga (Figura 4.4) com o objetivo de se observar a influência do retardo no "método 2".

O algoritmo apresentado (Figura 4.4) mostra os intervalos em que a carga sofrerá reparos. Nesta figura "C" representa a quantidade de vezes em que o processo deverá ser reativado. Essa quantidade será incrementada cada vez que o processo P2 sofrer uma interrupção.

<table>
<tr><th colspan="2">Carga < 300 ms</th><th>300 ms < Carga</th></tr>
<tr><th>C = 0</th><th>C > 0</th><td rowspan="2">Ativa P1;
Para P2;
C = C + 1</td></tr>
<tr><td>Ativa P1;
Ativa P2;</td><td>Ativa P1;
C = C –1;
Ativa P2;</td></tr>
</table>

Figura 4.4- Tratamento dos eventos do gerente de carga com o "método 1"

4.5 Conclusão

Os reinícios propiciam uma maneira eficiente de tratar alguns dos inconvenientes causados pela sobrecarga do servidor com a lentidão e a conseqüente indisponibilidade de serviços. Utilizando-se threads estes mecanismos de tratamento de processos (interrupções e inícios) oferecem maior velocidade. Desta forma os processos gerados pelas requisições são controlados por threads de acordo com os critérios os apresentados neste Capítulo.

Com base nessas premissas supracitadas (reinício, threads e critérios de carga), o mecanismo de interrupção e reinício se habilita

a solucionar problemas de capacidade de processamento do servidor tempo de execução. Para comprovação disso foram realizados experimentos a fim de se conhecer a viabilidade no emprego dessa técnica.

5. Experimentos e Resultados

Este capítulo contém os detalhes relativos à configuração e implementação do novo componente da plataforma WS-DSAC.

5.1.1 Configuração física

Foi utilizado um cluster composto por cinco computadores (Figura 3.1) com a seguinte configuração: Processador Intel Pentium IV, com CPU de 2267.63 Mhz e 512 de memória RAM, interconectados por uma rede local.

5.1.2 Configuração lógica

Foi utilizada a tecnologia Java 2 SDK, Standard Edition 5.0 [Java], servindo-se da API RMI (Java Remote Method Invocation) para invocações remotas de objetos. Como conteiner o Apache Tomcat 5.5.9 [Apache] sobre o kernel 2.6.8.1 da distribuição Linux Kurumin 5.0 [Morimoto, 2004] [Morimoto, 2006].

5.2. Descrição dos experimentos

Por ocasião dos experimentos foram injetadas requisições ininterruptas a cada dois segundos. Os experimentos foram realizados de acordo com os critérios comentados no tópico 4.4. Na Tabela 4.2 é demonstrada a classificação dos processos conforme o seu tempo de processamento.

5.3 Resultados dos experimentos

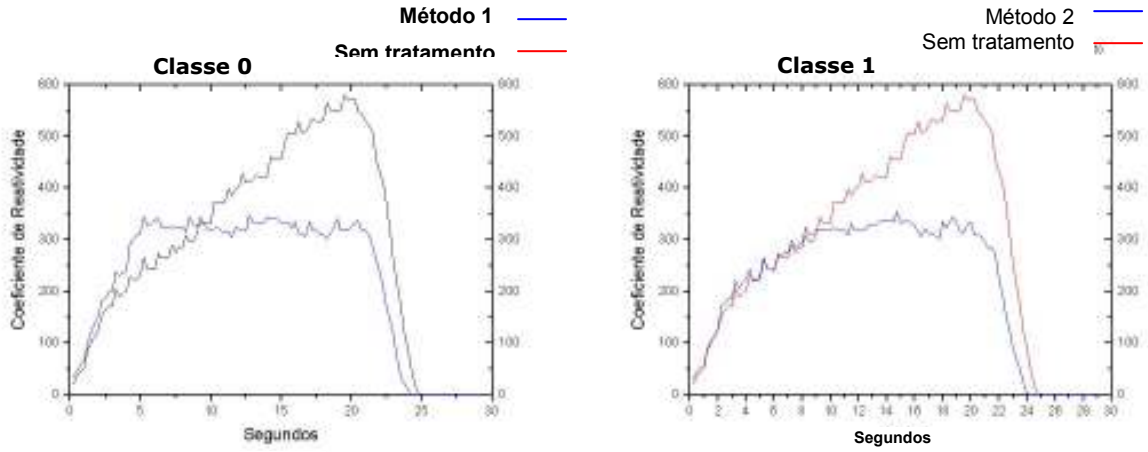
Por ocasião dos experimentos procurou-se observar o comportamento da carga dos servidores de acordo com os critérios já citados no Capítulo 4, com o objetivo de analisar a capacidade da técnica de interrupção e reinício sob diferentes níveis de carga.

Utilizou-se para isso um cluster composto por cinco computadores, sendo quatro nodos servidores e um nodo gerente (Figura 3.1).

Em todos os experimentos foram realizadas requisições, na intensidade de uma a cada dois segundos num período de 20 segundos com uma carga de 2.375 do tipo 2 (Tabela 4.2). Para efeito dos testes a carga gerada foi constituída de um servlet.

Nesta primeira fase dos experimentos procurou-se avaliar a viabilidade no uso da técnica de interrupção e reinício de requisições no tratamento da carga.

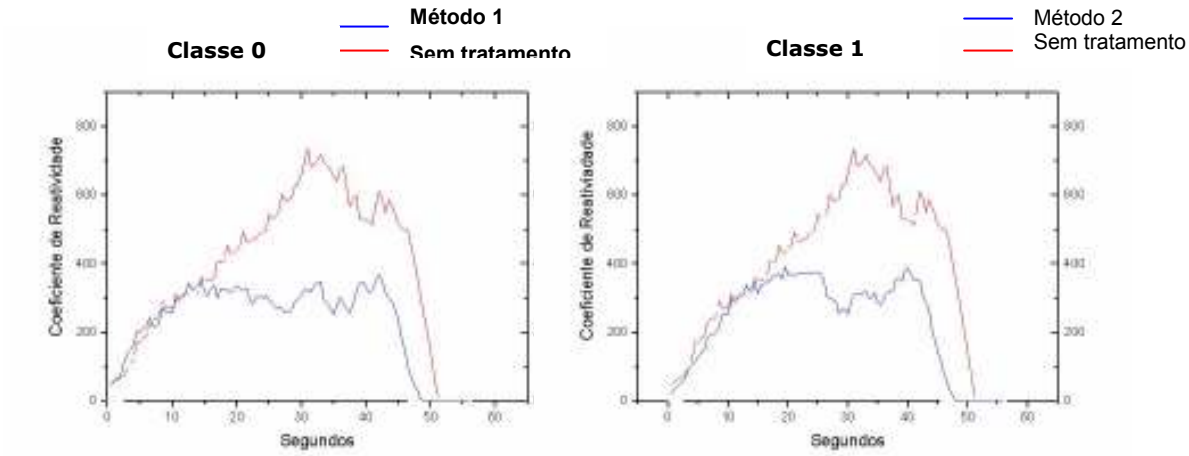
Na Figura 5.1 são apresentados os resultados com tratamento aplicando o método sem retardo na ativação do processo que exige maior tempo de processamento (a), denominado "método 1" (tópico 4.4 c), e com o retardo, rotulado como "método 2". Em ambos os métodos, foram aplicados o mesmo tipo de tratamento de carga, ou seja, interrupção e reinício de processos, sendo que na segunda abordagem além da suspensão do processo adicionou-se um pequeno retardo de 300 ms sobre o processo interrompido. Comprovou-se a eficiência dos dois métodos tanto no controle da carga dos nodos quanto no tempo de término da execução de todas as tarefas. Nota-se na Figura 5.1 que todas as requisições tratadas com o "método 1" (Figura 5.1 a) ou "método 2" (Figura 5.1 b) obtiveram um coeficiente de reatividade inferior de 600 ms. Significa que todas as requisições foram respondidas. Além disso, ambas tiveram um tempo de execução inferior ao servidor em que não houve tratamento da carga.



(a) (b)

Figura 5.1: Simulação com um processo de 2.375 ms

Na segunda fase dos testes estudou-se o comportamento da carga com alguns diferenciais do primeiro experimento. Foram empregados dois processos, um com o tempo de duração (execução) de 100ms e outro com o tempo de 2.654 ms. O processo de maior tempo de execução foi interrompido e reativado conforme os métodos 1 e 2 já comentados. Na Figura 5.2, nota-se comportamento semelhante em relação ao teste anterior que tende a uma estabilidade da carga, sendo confirmando o ganho de desempenho ao término das execuções dos processos com o "método 1"(Figura 5.2 a) e "método 2"(Figura 5.2 b).



(a) (b)

Figura 5.2: Simulação com processos de 100 ms e de 2.654 ms

Conforme a Tabela 5.1 podemos observar o resultado da simulação com processos de cargas do "tipo 1" e do "tipo 4".

Constata-se que o servidor sem tratamento da carga saturou, perdendo em torno de 19% das requisições (classe 0). Com o "método 2", o servidor não trabalhou saturado utilizando-se e todas as requisições foram atendidas, enquanto que aproximadamente 0,83% da requisições foram rejeitadas com o uso "método 1".

Tabela 5.1: Quantidade de ocorrências por carga

Sem tratamento	
carga	**%**
0 - 100	13,33%
100 - 200	40,83%
200 - 300	5,83%
300 - 400	5,83%
400 - 500	3,33%
500 - 600	11,67%
600 - 700	**7,50%**
700 - 800	**5,00%**
800 - 900	**2,50%**
900 - 1000	**4,17%**
	100%

Método 1	
Carga	**%**
0 - 50	30,00%
50 - 100	0,00%
100 - 150	6,67%
150 - 200	3,33%
200 - 250	16,67%
250 - 300	10,00%
300 - 350	19,17%
350 - 400	5,83%
400 - 450	1,67%
450 - 500	1,67%
500 - 550	4,17%
550 - 600	0,00%
600 - 650	**0,83%**
	100%

Método 2	
carga	**%**
0 - 50	30,83%
50 - 100	0,00%
100 - 150	6,67%
150 - 200	0,00%
200 - 250	20,00%
250 - 300	5,00%
300 - 350	9,17%
350 - 400	15,83%
400 - 450	0,83%
450 - 500	6,67%
500 - 550	4,17%
550 - 600	0,83%
	100%

A Tabela 5.2 demonstra outra simulação em que são comparados os resultados obtidos na aplicação dos métodos 1 e 2 sob uma carga com processamento de 2.375 segundos de duração na classe 1, constatando-se pouca diferença entre ambos. Porém mostram-se igualmente eficientes nos testes em relação ao método em que não se aplicou o tratamento da carga do servidor. Conforme a tabela 5.2 24,55% das requisições são rejeitadas enquanto que a aplicação de interrupção e reinício ("método 1" e "método 2") de processos não alcançam 1% das requisições.

Tabela 5.2: Comparação dos resultados usando"método 1" e o "´método2"

Classe 0	Porcentagem de requisição (%)		
Estado do servidor	sem tratamento	método 1	método 2
Estado compartilhado	32,73	59,09	60,91
Estado exclusivo	42,73	40,91	39,09
Estado saturado	24,55	0,00	0,91
Total	100,00	100,00	100,00

Em outra situação foram utilizados dois processos a cada dois segundos, um do "tipo 1" (Tabela 4.2) e outro com duração de 21.213 ms em uma "classe 0" . Inicialmente a sobrecarga no servido sem tratamento atinge valores críticos (aproximadamente dois segundos) até estabilizar-se em um pico 690 ms e finalmente terminar a execução dos processos.

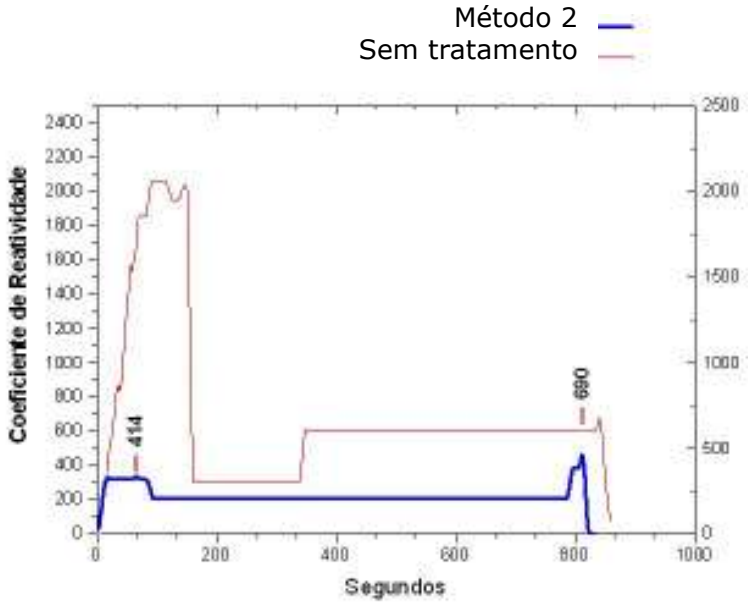


Figura 5.3: Simulação com processos de 100 ms e de 21.213 ms

A aplicação do mecanismo de reinícios mostra-se mais ainda eficiente tanto no ganho de desempenho, pois seus processos findam após 813 segundos (contra 854 segundos da curva que representa o comportamento da carga sem tratamento) quanto em requisições respondidas. Este comportamento é analisado no próximo tópico.

5.4 Análise dos Resultados

Através dos experimentos pode-se observar primeiramente que o emprego da técnica de interrupção e reinício permite minimizar atrasos gerados tanto por paradas curtas quanto por paradas longas, pois quando um único thread pára, as demais instruções de outros threads continuam a ser executados segundo uma dada prioridade.

Outra observação está relacionada ao tempo médio das tarefas (requisições) e a ordem de prioridade dessas tarefas executadas pelo processador de cada nodo. O mecanismo empregado faz com que o tempo médio de ocupação da CPU seja otimizado mesmo com as interrupções (Tabela 5.3). Isto porque as interrupções de threads ocorrem ao nível do usuário e não ao nível de kernel [Silberchatz, 2000]. Pode-se notar ainda pelos resultados apresentados na seção anterior (5.3), que o tempo de execução total em todos os casos foi reduzido.

Tabela 5.3: Estatística da média do coeficiente de reatividade

	Tempo de Computação (ms)	**Experimento**	**Média (ms)**	Desvio Padrão
Classe 0	2.375	Sem reinícios	351,7636	163,11039
		Com reinícios	172,8182	43,42302

Classe 1		Sem reinícios	351,375	119,3348
		Com reinícios	189,45	23,52542
Classe 0	2.654	Sem reinícios	1214,748	138,37931
		Com reinícios	201,7375	57,69557
Classe 1		Sem reinícios	521,1573	194,59006
		Com reinícios	190,6237	36,01446

Quando o processo esgota sua fatia de tempo (quantum), é interrompido e colocado no fim da fila de processos. Isto significa que um processo pode ser posto diversas vezes na fila de execução pelo fato de seu quantum expirar e desta forma atrasar processos com menor tempo de execução e conseqüentemente aumentando o tempo de espera na fila [Ribeiro, 2005]. Esta situação leva à diminuição do número de processos executados por segundo. Além disso, alternar a CPU para outro processo requer salvar o estado do processo antigo e carregar o estado salvo do novo processo. Isto força o kernel a carregar os contextos diversas vezes e a aumentar seu overhead. A justificativa disso se deve ao fato dos threads de kernel serem mais lentos para criar e gerenciar do que os threads de usuário.

Os threads de usuário são suportados acima do kernel e são implementados por uma biblioteca de threads a nível do usuário [Silberchatz, 2000]. A biblioteca fornece suporte à criação, escalonamento e gerência de threads através da interrupção e reinício de processos no espaço de usuário, sem a necessidade da intervenção do kernel. Os experimentos apresentados comprovam que o uso de interrupções sobre as requisições do usuário (threads do usuário) resultaram num tempo de processamento menor em relação à execução das mesmas tarefas sem tratamento.

Ademais, as threads reduzem o gasto desnecessário de memória e processamento, uma vez que o processo principal e os

processos leves compartilham, além do segmento de código, o segmento de dados e espaço de endereçamento. Elas possuem apenas sua própria pilha de execução e o seu próprio programa "counter".

No caso do UNIX/LINUX, a criação de um processo leve difere da criação de um processo de maior esforço computacional nos seguintes aspectos [Ribeiro, 2005]:

- Criar e destruir threads exige menor tempo e controle do sistema operacional;
- As threads usam os segmentos de código e dados do processo de maior tempo de execução;
- As operações de comunicação e sincronização entre processos muito eficazes.

Por ocasião dos experimentos ficou comprovada que a taxa de requisições rejeitadas foi reduzida em todos os casos. Isto é resultado da diminuição do tempo de resposta. Com relação aos "método 1" e "método 2" implementados, observou-se a eficiência dos dois por utilizarem o mesmo mecanismo de tratamento (interrupção e reinício).

6. Conclusão

6.1 Contribuições e Resultados Alcançados

Neste trabalho foi proposto um mecanismo de gerência de carga para clusters de servidores Web.

Na implementação desse mecanismo foi apresentada uma técnica denominada interrupção e reinício de processos, que consiste no cancelamento de processos para em seguida reativá-los de acordo com as condições da carga da máquina. Esta técnica foi implementada com a finalidade de gerenciar processos em servidores Web através de um gerenciamento de tarefas, priorizando as aquelas de menor tempo de processamento.

Entre as contribuições deste trabalho pode-se destacar os seguintes:

- O mecanismo de interrupção e reinício mostrou-se uma ferramenta eficiente como solução pró-ativa no controle de sobrecarga em servidores Web;
- Utilizando o coeficiente de reatividade como parâmetro de medição de carga, verificou-se que a técnica de interrupção e reinício obteve um menor tempo de resposta do servidor;
- Este gerenciamento fez diminuir o número de solicitações rejeitadas no WS-DSAC;
- A técnica garante uma boa disponibilidade nos serviços prioritários decorrente de um melhor gerenciamento dos processos;
- Como conseqüência imediata da redução do tempo de resposta foi possível melhorar a qualidade do serviço.

Gerência baseada no mapeamento das tarefas coordenada por grupos de threads.

6.2 Trabalhos Futuros

Para finalizar sugere-se como contribuições futuras na continuidade deste trabalho a implementação de um dispositivo que bloqueie num curto espaço de tempo o acesso aos recursos de cada nodo servidor. Desta maneira impedir-se-ia que sequer o processo provindo de uma requisição fosse criado, evitando-se que o sistema tivesse alguma perda com o tempo nas interrupções de processos a nível de usuário. Essa proposta de trabalho futuro visa estudar a eficiência ou não dessa abordagem em relação à solução apresentada nesta dissertação.

Também se sugere a análise e implementação de novas políticas de escalonamento que se adaptem dinamicamente conforme a carga do servidor para otimização de sua capacidade de processamento.

BIBLIOGRAFIA

ANTONOPOULOS, D. M.; HOUSIADAS, C., ”Moderator temperature coefficient of reactivity in Pressurized Water Reactors: theoretical investigation and numerical simulations”. Nuclear Science and Engineering, 1999.

APACHE JAKARTA; "The Apache Jakarta Project". Disponível na internet: http://jakarta.apache.org/tomcat/index.html. (Data da consulta: 23/2/2006).

ASM, Disponível na internet: http://www.asm-usa.com/software/PAB3D/cluster/. (Data da consulta: 23/2/2006).

AWDUCHE, D.; "Requirements for Traffic Engineering over MPLS", RFC2702, Setembro 1999.

BREWER, E.; “Lessons from giant-scale services”. IEEE Internet Computing, 5(4):46-55, July 2001.

CANDEA G.; FOX A.; “Recursive Restartability: “Turning the Reboot Sledgehammer in to a Scalpel”, Appears in Proceeding soft he 8th Workshopon Hot Topics in Operating Systems(HotOS-VIII), May 2001.

CANDEA, G., CUTLER J.; FOX, A.; DOSHI R.; GARG , G. P.; “Reducing Recovery Time in a Small Recursively Restartable System”. International Conference on Dependable Systems and Networks(DSN), Washington, D. C., June2002.

CANDEA, G.; DELGADO M.; CHEN M.; FOX A.; "Automatic Failure Path Inference: A generic Introspection Technique for Internet Applications", 3rdIEEE Workshopon Internet Applications (WIAPP), San Jose, CA, June 2003.

CANDEA, G.; KAWAMOTO, S.; FUJIKI, Y.; FRIEDMAN, G., FOX A.; “Microreboot-A Technique for Cheap Recovery”. Proc.6th Symposiumon Operating Systems Designand Implementation(OSDI), San Francisco, CA, December 2004.

CANDEA, G.; BROWN, A.; FOX, A.; PATTERSON, D., “Recovery Oriented Computing:Building Multi-Tier Dependability”. IEEE Computer, Vol.37, No.11, November 2004.

CARPENTER, B. E.; NICHOLS, K.; “Differentiated services in the Internet”, Proceedings of the IEEE , Vol. 90, Issue: 9 , Sept. 2002, Pages:1479-1494, 2002.

DIGIORGIO A.; "The smart shipis", note nough Naval Institute Proceedings, 124(6), June 1998.

Disponível na internet: www.estadao.com.br/tecnologia/informatica/2003/mar/31/280.htm. (Data da consulta: 23/2/2006).

HENNESSY J.; PATTERSON D.; "Computer Architecture: Aquatitative Approach", San Francisco, CA, 3rd edition, 2002.

IEEE Computer Society Task Force on Cluster Computing, http://www.ieeetfcc.org/.

ISO/IECDIS13236,"Information Technology-Quality of Service-Framework", ISO/OSI/ODP, Julho 1995.

JALOTE, P.; "Fault Tolerance in Distributed Systems". Englewood Cliffs, PTR PrenticeHall, 1994.

JAVA; Disponível na internet: http://java.sun.com/docs. (Data da consulta: 23/2/2006).

LINUX JOURNAL; Disponível na internet: http://www.linuxjournal.com/article/2494. (Data da consulta: 23/2/2006).

KEMBEL, R. W. ; "The Fibre Channel Consultant: A Comprehensive Introduction", Northwest Learning Associates, 1998.page 8.

MORIMOTO, C. E.; "Linux:Entendendo o sistema-Guia Prático", Em versão digital. Disponível em: http://www.guiadohardware.net/livros/kurumin. (Data da consulta: 23/2/2006).

MORIMOTO, C. E.; "Kurumin:desvendando seus segredos O GUIA OFICIAL", Editora: AltaBooks, 2004.

LINUX, Disponível na internet: http://www.linuxdevcenter.com/pub/a/linux/2004/12/29/lnxclstrs_10.html. (Data da consulta: 23/2/2006).

LINUX MAGAZINE, Disponível na internet: http://www.linux-magazin.de/Artikel/ausgabe/1999/02/Cluster/cluster.html. (Data da consulta: 23/2/2006).

MELLO, R. F.; ”Proposta e avaliação de desempenho de um algoritmo de balanceamento de carga para ambientes distribuídos heterogêneos e escaláveis. Tese (Doutorado), Escola de engenharia de São Carlos (EESC), São Carlos, São Paulo, Brasil, 2003.

OLEJNIK, R.; BOUCHI, A.; TOURSEL, B.; "An Object observation for a Java Adaptative Distributed Application platform", IEEE PARELEC'02,22-25.

PECW: Disponível na internet: www3.cefetsc.edu.br/julio/paginas/pesquisa/PECW.pdf. (Data da consulta: 23/2/2006).

PITANGA, M.; "Construindo supercomputadores em Linux", Ed. Brasport, 2002.

REEVES G.; “What really happened on Mars ? RISKS-19.49”, Jan.1998.

RIBEIRO, U.; Sistemas Distribuídos : Desenvolvendo Aplicações de Alta Performace no Linux”, Axcel, 2005

Sum Microsystems; "Java Remote Method Invocation", Disponivel na Internet: http://java.sun.com/products/jdk/rmi/. (Data da consulta: 23/2/2006).

SENGER, L. J.; "Escalonamento de processos: uma abordagem dinâmica e incremental para a exploração de características de aplicações paralelas", Tese de Doutorado em Ciência da Computação e Matemática Computacional (S.Carlos)- Universidade de São Paulo, 2004.

SERRA A.; GAÏTI D., BARROSO G.; RAMOS R., BOUDY, J.; "Uma Plataforma Distribuída com Balanceamento de Cargas para Servidores Web Baseada na Diferenciação de Serviços",Proceedings of the SEMISH-SBC, pp.93-105, 2004a.

SERRA, A.; GAÏTI, D.; BARROSO;G.; BOUDY, J.; "Aload-balancing Distributed Platform based on Differentiated Services for a Telecare Application", Proceeding sof the IEEE International Conference on Control Application, Taipei-Taiwan, September2-4, Pag.69-74, 2004b.

SERRA, A.; GAÏTI, D.; BARROSO, G.; RAMOS, R.; BOUDY, J.; "WS-DSAC: Um Mecanismo de Controle de Admissão baseado na Diferenciação de Serviços para uma Plataforma Distribuída de Servidores Web", Portuguese track in the 3rd International Information and

Telecommunication Technologies Symposium, São Carlos/SP-Brazil , 6-9 december, 2004.

SERRA, A.; GAÏTI, D.; BOUDY, J.; BARROSO, G.; RAMOS, R.; "WS-DSAC: Na Admission Controland Load balancing Mechanism to Assure QoS Differentiation on Web Servers Clusters", in Proceeding of IFIP/IEEE International Symposium on Integrated Network Management (IM2005), 15-19 may 2005, Nice-France, 2005a.

SERRA, A.; GAÏTI, D.; CARDOSO, K.; BARROSO, G.; RAMOS, R.; "Controle de Admissão e Diferenciação de Serviços em Clusters de Servidores Web",in Proceedings of Simpósio Brasileiro de Redes de Computadores-SBRC2005, Fortaleza/CE-Brazil, may 2005b.

SILBERCHATZ, A.; "Sistemas Operacionais: conceitos e aplicações", 4ª edição, Editora Campus, 2000.

THREADS; Disponível na internet: http://java.sun.com/docs/books/tutorial/essential/TOC.html#threads.