UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMATICA

CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

DANIELA MEDEIROS CEDRO

GESTÃO DA QOS EM ARQUITETURAS DE GRADES COMPUTACIONAIS ORIENTADAS A SERVIÇOS

FORTALEZA – CEARÁ
AGOSTO/2010

UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMATICA

CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

GESTÃO DA QOS EM ARQUITETURAS DE GRADES COMPUTACIONAIS ORIENTADAS A SERVIÇOS

Autor

Daniela Medeiros Cedro

Orientador

Prof. Dr. José Neuman de Souza

Co-Orientador

Prof. Dr. Antonio de Barros Serra

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará, como parte dos requisitos exigidos para obtenção do grau de **Mestre em Engenharia de Teleinformática**.

FORTALEZA – CEARÁ

AGOSTO/2010

DANIELA MEDEIROS CEDRO

GESTÃO DA QOS EM ARQUITETURAS DE GRADES COMPUTACIONAIS ORIENTADAS A SERVIÇOS

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia de Teleinformática e aprovada em sua forma final pelo programa de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará.

_____

Daniela Medeiros Cedro

Banca Examinadora

_____

Prof. Dr. José Neuman de Souza

Orientador

_____

Prof. Dr. Antonio de Barros Serra

Co-Orientador

_____

Prof. Dr. Antonio Clécio Fontenelle Thomaz
Universidade Estadual do Ceará

_____

Prof. Dr. Giovanni Cordeiro Barros
Universidade Federal do Ceará

_____

Prof. Dr. Raimundo José de Araújo Macêdo
Universidade Federal da Bahia

FORTALEZA – CEARÁ

AGOSTO/2010

Dedico este trabalho aos meus Anjos da Guarda: tia Lena e vó Toinha

Agradecimentos

Aos meus filhos e ao meu marido que tiveram que suportar todos minhas angustias e irritações, me compreendendo, apoiando e incentivando durante todo caminhar do mestrado.

Aos professores José Neuman de Souza, Antonio de Barros Serra pela valorosa orientação.

Aos professores Giovanni Cordeiro Barroso e Rossana Maria de Castro Andrade pelo apoio e exemplos de garra e profissionalismo.

Aos meus colegas de curso: Aparecida Prado, Arisson Leal, Igor Valente, Thialy Vercelly, André Nunes e Geordano Gadelha pelas preciosas trocas de conhecimentos que ajudaram na minha formação.

Ao colega Leonardo Cardoso que contribuiu bastante com a implementação da plataforma G-DSAC e a minha amiga Viviane que me ensinou a ver a alegria mesmo em momentos difíceis.

Resumo

A crescente disponibilização de serviços através da Internet vem impondo uma demanda cada vez maior por recursos de processamento no lado servidor favorecendo a utilização dos Clusters de Computadores e das Grades Computacionais. Em paralelo, a engenharia de software traz novos paradigmas, como a Orientação a Serviços, que impõem novos desafios a serem tratados pelos fornecedores de serviços. A convergência destes fatores deu origem às Arquiteturas de Grades Computacionais Orientadas a Serviços. Neste trabalho é apresentada uma proposta de arquitetura em grades computacionais orientada a serviços, denominada G-DSAC (*Grid – DiffServ Admission Control*), que trata de aspectos ligados à QoS (*Quality of Service*) e a diferenciação de serviços. A arquitetura G-DSAC é uma extensão da arquitetura WS-DSAC (*Web Servers – DiffServ AdmissionControl*). Esta extensão compreende a concepção de uma solução voltada para grades computacionais que é capaz de garantir os SLAs (*Service Level Agreements*) estabelecidos com os consumidores de serviços utilizando de forma otimizada os recursos de processamento disponibilizados na grade. A solução permite ainda a diferenciação de serviços no que diz respeito aos tempos de resposta oferecidos aos clientes, usuários finais e serviços consumidores. A nova arquitetura proposta introduz um bloco de funcionalidades em uma plataforma de grade computacional orientada a serviços formada por multclusters. Esse bloco permite a publicação e localização de serviços, autenticação e classificação de requisições e o escalonamento das mesmas dentro da grade de acordo com a classe de serviço a qual pertencem. Foi também implementado um protótipo que permitiu a realização de experimentos em uma plataforma real de testes visando avaliar a capacidade da solução em atingir os objetivos por ela propostos.

Abstract

The increasing deployment of services on the Internet is augmenting the demand for processing resources on the server side encouraging the use of Clusters of Computers and Grid Computing. At the same time, the engineering software brings new paradigms such as Service Orientation, which impose new challenges to be addressed by service providers. The convergence of these factors led Architectures for Service Oriented Grid Computing. This work presents an architecture proposed for service oriented grid computing, called G-DSAC (Grid - DiffServ Admission Control), which deals with aspects related to QoS (Quality of Service) and service differentiation. The G-DSAC architecture is an extension of WS-DSAC architecture (*Web* Servers - AdmissionControl DiffServ). This extension includes design of a Service Oriented Grid Computing Architecture which is able to ensure SLAs (Service Level Agreements) established with consumers using effectively the resources available in the grid. The solution also provides service differentiation in relation to response times offered to customers, end users and service consumers. The proposed new architecture introduces a block of functionality in a service oriented grid computing platform composited by Multclusters. This block allows publishing and localization of services, authentication and classification of requests and scheduling of those within the grid according to the service class they belong. It was also implemented a prototype that allowed the realization of experiments in a testing platform to evaluate the solution's ability to achieve the objectives proposed by the new architecture.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

SOA	Service-Oriented Architecture
QoS	Quality of Service
OV	Organização Virtual
OGSA	Open Grid Services Architecture
SLA	Service Level Agreement
MAN	Metropolitan Area Network
WAN	Wide Area Network
LAN	Local Area Network
IP	Internet Protocol
VoIP	Voice over Internet Protocol
TI	Tecnologia da Informação
URL	Uniform Resource Locator
LBMS	Load Balancing Mechanism Specifications
HTTP	HyperText Transfer Protocol Secure
CRM	Cluster Resource Manager
RG	Request Gateway
MOG	Message Object Gateway
GRM	Global Resource Manager
WWW	World Wide Web
MPLS	Multiprotocol Label Switching
UDDI	Universal Description, Discovery and Integration
WSDL	*Web Service Description Language*
CPU	Central Processing Unit
XML	eXtensible Markup Language
WSDD	Web Service Deployment Descriptor
API	Application Programming Interface
SGBD	Sistema Gerenciador de Banco de Dados

SQL	Structure Query Language
RMI	Remote Method Invocation
JDK	Java Development Kit
IDE	Integrated Development Environment

I
Introdução

1.1. Motivação

Com a proliferação de serviços oferecidos através da *Web*, os clusters de servidores *Web* [MORRISON, 2003] se tornaram uma alternativa importante para a melhoria na qualidade dos serviços ofertados através da Internet, principalmente no que diz respeito ao tempo de resposta no atendimento de requisições dos usuários. *Clusters* de computadores permitem a interligação em rede de um conjunto de equipamentos aumentando a disponibilidade de um sistema bem como o seu desempenho [BUYYA, 1999].

Com o avanço dos meios de comunicação e o conseqüente aumento nas taxas de transmissão, principalmente devido ao uso crescente das fibras ópticas, a idéia de compartilhar o poder de processamento tomou uma dimensão ainda maior fazendo emergir a computação em grade, onde recursos geograficamente distribuídos podem ser compartilhados com uma gestão realizada através de interfaces e protocolos padronizados, abertos e de propósitos gerais [FOSTER; KELSSELMAN, 2004] [NASSIF; NOGUEIRA, 2007].

Em paralelo a essa evolução, que passa pelos clusters e chega às grades computacionais, houve um avanço nas tecnologias utilizadas para implementar e disponibilizar serviços oferecidos através da Internet. Assim, as páginas estáticas disponibilizadas na WWW foram progressivamente sendo substituídas por páginas dinâmicas (Serviços Web) implementadas em diferentes tecnologias (CGIs, Servlets, JSPs, ASP, etc.) [CEDRO, 2004]. Hoje, novos paradigmas da Engenharia de Software estão fundamentados na idéia da orientação a serviços, que têm como premissa básica a disponibilização de serviços de forma distribuída permitindo o reuso e a orquestração dos mesmos [KEEN et al., 2004]. Isto também é devido aos avanços dos meios de transmissão que minimizaram os custos de comunicação nas trocas de mensagens

[TANENBAUM; STEEN, 2007]. Neste contexto, surgiu um novo conceito, o de *WEB Services*, que permitiu a implementação de serviços com interoperabilidade entre diferentes plataformas tomando como base o uso de protocolos padrões na Web (HTTP, XML, SOAP,...) [PUTTE et al, 2004].

A evolução paralela da computação em grade e da orientação a serviços fez surgir uma nova área de estudos onde se encontram as arquiteturas em grade orientadas a serviços dando origem a padrões como o OGSA (*Open Grid Services Architecture*) [FOSTER et al, 2002]. Nestas arquiteturas, as vantagens oferecidas por um ambiente em grade (disponibilidade, poder de processamento, etc.) são aliadas as vantagens da orientação a serviços (reuso, orquestração, extensão, etc.).

Um ponto crítico a ser tratado dentro das arquiteturas em grade orientadas a serviços é a questão da gestão da QoS (*Quality of Service*) oferecida aos usuários da grade. Levando em consideração que o ponto de sobrecarga (ponto de estrangulamento) vem se localizando cada vez mais nos recursos de processamento, duas possíveis soluções largamente utilizadas para lidar com este problema e aumentar a QoS do usuário final são: a distribuição equilibrada da carga imposta pelos serviços dentro da grade e o uso de mecanismos de controle de admissão para evitar a sobrecarga ou possível "crash" de servidores [CASTRO; DWYER; RUMSEWICZ, 1999] [CHERKASOVA; PHAAL, 2002] [ABDELZAHER; SHIN; BHATTI, 2002].

Por outro lado, usuários com expectativas diferentes em relação à QoS desejada, mesmo podendo pagar mais caro pelos serviços utilizados, são forçados a compartilhar de forma igual os recursos de processamento disponibilizados para o provimento de um serviço na WWW. No nível de redes baseadas no protocolo IP, mecanismos como o DiffServ (*Differentiated Services*)[CARPENTER; NICHOLS, 2002] ou MPLS (*Multiprotocol Label Switching*)[ROSEN et al., 2001], permitem a diferenciação de serviços agrupando-os em classes e garantindo a cada classe certos parâmetros de QoS (perdas, retardo, variação do retardo). No nível de aplicação, para permitir que fornecedores de serviços possam diferenciar a QoS oferecida aos seus clientes, é necessária a utilização de mecanismos que permitam o uso diferenciado dos recursos de processamento disponíveis. Um exemplo deste tipo de mecanismo é o WS-DSAC [SERRA, 2005].

O WS-DSAC é uma arquitetura que implementa um mecanismo de controle de admissão e de balanceamento de cargas concebido para clusters de servidores Web. O mecanismo permite a diferenciação de serviços através do agrupamento de requisições em classes diferentes de serviços. Um parâmetro de QoS denominado "Coeficiente de Reatividade" [SERRA; BARROSO; BOUDY, 2004] (diretamente relacionado ao tempo de resposta do cliente) é associado a cada classe de serviços pré-estabelecida. O mecanismo 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.

No entanto, quando a arquitetura WS-DSAC foi concebida, levou em consideração ambientes baseados em clusters de servidores Web, ou seja, a arquitetura WS-DSAC não trata conceitos importantes ligados a orientação a serviços nem a computação em grade, não se adequando aos novos paradigmas da computação distribuída, notadamente as Arquiteturas em Grade Orientadas a Serviços.

Dentro deste contexto, este documento apresenta uma proposta de trabalho que trata de aspectos ligados à QoS (Quality of Service) em Arquiteturas em Grades Computacionais Orientadas a Serviços.

1.2. Objetivo

Este trabalho teve como objetivo principal estender a Arquitetura WS-DSAC (*Web Service – DiffServ Admission Control*) [SERRA, 2005] (ver Capítulo III), inicialmente concebida com foco nas arquiteturas em *cluster*, para o mundo da orientação a serviços utilizando grades computacionais. Esta extensão, denominada G-DSAC (*Grid DiffServ Admission Control*), compreende a concepção de uma solução voltada para grades computacionais [NASSIF; NOGUEIRA, 2007] que seja capaz de garantir os SLAs (*Service Level Agreements*) estabelecidos com os clientes utilizando de forma otimizada os recursos de processamento disponibilizados na grade. A solução proposta permite ainda a diferenciação de serviços no que diz respeito aos tempos de resposta oferecidos aos usuários finais e aos serviços consumidores.

1.2.1. Objetivos específicos

Podem ser citados como objetivos específicos deste trabalho:

- A concepção de uma solução arquitetural para um ambiente de grades computacionais orientada a serviços capaz de prover balanceamento de cargas na grade e a diferenciação de serviços no que diz respeito ao tempo de resposta das requisições;

- A extensão da arquitetura WS-DSAC [SERRA, 2005] para adequá-la a nova solução arquitetural concebida que trata aspectos relacionados às grades computacionais orientadas a serviços;

- A adaptação do mecanismo WS-DSAC [SERRA, 2005], concebido inicialmente para clusters de servidores Web, visando adequá-lo ao uso em um ambiente de Grade Computacional Orientada a Serviços;

- A implementação de um protótipo da arquitetura proposta com o objetivo de disponibilizar um ambiente real de testes para avaliação da solução;

- A realização de experimentos visando avaliar a capacidade da solução proposta em balancear cargas e prover a diferenciação de serviços utilizando de forma eficaz os recursos disponíveis na grade.

1.3. Estrutura do Documento

Este documento está organizado da seguinte forma:

No capítulo II, são contextualizados temas que servirão como base para o melhor entendimento dos capítulos seguintes. São abordados: as Arquiteturas Orientadas a Serviços (SOA); os ambientes de computação em grade; as grades computacionais baseadas em *MultiClusters*; trabalhos científicos relacionados ao tema da QoS e da diferenciação de serviços; a gestão da QoS em ambientes distribuídos; os ambientes em grade *Globus*, *Ourgrid* e *EuroGrid* e a computação nas nuvens.

No capítulo III, é apresentada a plataforma WS-DSAC cuja concepção foi voltada para ambientes de clusters de servidores Web. São descritos a arquitetura e o mecanismo WS-DSAC com os blocos e os elementos que os compõem assim como os parâmetros e o algoritmo que integram o mecanismo.

No capítulo IV, são abordados aspectos relacionados à nova arquitetura proposta, denominada G-DSAC, que foi concebida com foco nas arquiteturas em multiclusters para o mundo da orientação a serviços utilizando grades computacionais. O capítulo aborda aspectos relacionados à nova plataforma proposta, iniciado pela declaração do problema a ser tratado, seguida da descrição da arquitetura e de seus elementos, do mecanismo G-DSAC e dos trabalhos relacionados.

No capítulo V, são apresentadas as tecnologias utilizadas na implementação da plataforma G-DSAC, os passos executados para o desenvolvimento do protótipo com as funcionalidades de cada elemento, o ambiente de teste e sua configuração e finalmente os experimentos realizados com os resultados obtidos.

No capítulo VI, são apresentadas as conclusões do trabalho incluindo as considerações finais, as contribuições, as limitações e os trabalhos futuros.

II
Computação Distribuída e QoS

2.1. Introdução

Neste capítulo serão abordados assuntos relacionados com o trabalho aqui apresentado de forma a facilitar a compreensão do contexto dentro do qual o mesmo está inserido e fornecer conceitos importantes que facilitarão a leitura dos capítulos seguintes.

Inicialmente, serão apresentados conceitos relacionados às Arquiteturas Orientadas a Serviços (SOA) e aos ambientes de computação em grade. Em seguida, será abordado o tema das Grades Computacionais baseadas em *MultiClusters*, alguns trabalhos científicos desenvolvidos sobre este tema serão destacados e o problema da gestão da Qualidade de Serviço em Ambientes Distribuídos serão discutidos.

2.2. Arquitetura Orientada a Serviços e as Grades Computacionais

A computação em grade surgiu da idéia do compartilhamento de recursos ociosos em uma rede geograficamente distribuída para se obter um maior poder de processamento e armazenamento além de permitir o uso compartilhado de diferentes recursos computacionais.

Foster e Kesselman definiram Grade Computacional como um sistema que coordena recursos distribuídos usando interfaces e protocolos padronizados, abertos e de propósitos gerais para entregar serviços de qualidade não trivial [FOSTER; KELSSELMAN, 2004].

Os principais elementos dessa definição são:

- ✓ Coordena recursos distribuídos;
- ✓ Usar interfaces e protocolos padronizados, abertos e de propósitos gerais;

- ✓ Entregar serviços de qualidade não trivial.

A coordenação é realizada em recursos e usuários que estão sob diferentes domínios e aborda temas de segurança, política, faturamento, afiliação, entre outros. Os protocolos e interfaces padronizados e abertos são constituídos através de propósitos múltiplos, relacionados com autenticação, autorização, descoberta de recursos e acesso a recursos. Por fim, a qualidade está relacionada com prover serviços variados e atender as demandas complexas.

A arquitetura da Grade Computacional é composta por quatro camadas (Figura 2.1). São elas: Aplicação; Cooperação ou Serviços Coletivos; Conectividade e Recursos; Construção (também conhecida como Fábrica).

Figura 2.1: ***Arquitetura de uma Grade Computacional*** **[NASSIF; NOGUEIRA, 2007]**

Aplicação: A camada de aplicações na arquitetura de grades computacionais compreende as aplicações do usuário que operam dentro de um ambiente de uma Organização Virtual.

Cooperação: A camada de serviços coletivos endereça problemas de descoberta, seleção e alocação de recursos, segurança, política e contabilização.

Conectividade e Recursos: A camada de conectividade e recursos define os protocolos de comunicação e de autenticação para transações específicas de grades computacionais. Esta camada é construída sobre os protocolos de comunicação e de autenticação da camada de

conectividade e seu papel é definir protocolos para negociação, inicialização, monitoração, controle, contabilização e faturamento de operações compartilhadas em recursos individuais.

Construção ou Fábrica: A camada de construção compreende os recursos para os quais os acessos compartilhados são mediados pelos protocolos da grade computacional.

Um ambiente de *cluster* é constituído de um sistema formado por hardware e software conectados em um local apenas, servindo a usuários que estão trabalhando somente em um projeto, usado exclusivamente para resolver os problemas computacionais de uma determinada organização. Por outro lado, uma grade presta serviços de uma forma geograficamente distribuída. Em um *cluster*, os recursos são gerenciados por uma entidade central, e os computadores agem como se fossem um único dispositivo. Nas configurações em grade, cada "organização virtual" faz o gerenciamento de seus recursos não tendo a visão de uma imagem única do sistema. Ou seja, o usuário tem consciência dos diversos serviços disponíveis e que deverá requisitá-los para sua utilização. Portanto, as grades são mais heterogêneas, complexas e distribuídas.

Uma Arquitetura Orientada a Serviços (*SOA*) possui como seu componente fundamental o conceito de serviços. Este conceito pode ser definido tanto como "componente" como "contrato". Se o serviço for compreendido como uma definição ou descrição de ações a serem executadas, um serviço é um contrato, contudo, se o serviço for definido como implementação direta dessas ações, um serviço é um componente [KEEN et al, 2004].

Algumas características são comumente identificadas na literatura:

- ✓ Reuso "Caixa-preta";
- ✓ Distribuição;
- ✓ Heterogeneidade Ambiental;
- ✓ Composição;
- ✓ Coordenação;
- ✓ Dinamismo e Adaptabilidade;
- ✓ Estado;
- ✓ Sincronia;
- ✓ Robustez de Protocolos.

A orientação a serviços e a computação em grade são tecnologias que se complementam. Para permitir que se tire o melhor destas duas tecnologias é necessária uma reflexão para se conceber arquiteturas que permitam a gestão de recursos de processamento e que implementem funcionalidades que dêem suporte a utilização da orientação a serviços em ambientes em grade computacional.

Além de prover serviços ligados à publicação de recursos, disponibilidade e balanceamento de cargas, a arquitetura proposta neste trabalho e que será apresentada nos capítulos seguintes também se propõe a prover mecanismos de apoio a gestão da qualidade de serviço (QoS) ofertada aos diversos usuários da grade computacional.

2.3. Grades Computacionais baseadas em Sistemas de *Multicluster*

Tendo em vista a existência de centros de pesquisa e instituições em geral, que possuíam uma infraestrutura computacional de múltiplos *clusters* que não aderiram à computação em grade de forma a aumentar o seu poder de processamento, armazenamento, entre outros e tendo o conhecimento de que diversos *clusters* possuíam seus funcionamentos isolados por decorrência de aquisições em períodos diferentes de tempo, iniciaram-se relevantes trabalhos de pesquisa com o objetivo de fazer a integração desses *clusters* como se formassem uma máquina exclusiva denominada pelos pesquisadores como sistemas de *Multiclusters*.

O sistema de *Multicluster* consiste em uma integração de *clusters* heterogêneos e independentes permitindo a construção de uma arquitetura paralela por meio de uma interface única [BARRETO; ÁVILA; NAVAUX, 2000]. A vantagem do *MultiCluster* é a possibilidade de ser configurado de acordo com as regras de uma determinada comunidade, como uma Organização Virtual (OV) pertencente a uma grade computacional.

Este conceito foi introduzido em 2000 através de um trabalho de pesquisa denominado "Projeto *MultiCluster*" [BARRETO; ÁVILA; NAVAUX, 2000] em que a idéia principal era permitir a interconexão de agregados baseados em diferentes tecnologias de comunicação. Desde então, outras pesquisas foram desenvolvidas, tais como:

- **Um modelo para a concepção de Grades Computacionais baseadas em *Clusters* [BARRETO; NAVAUX, 2003]:**

  Apresenta um modelo *MultiCluster* baseado no trabalho de pesquisa de Barreto [Barreto et al, 2000] e tem como objetivo proporcionar um estudo específico de uma grade computacional que utiliza recursos exclusivamente de *clusters* de alto desempenho e concluí apresentando uma validação por meio de um protótipo de integração de *clusters* que realizou as devidas considerações de acordo com as tecnologias utilizadas e demonstrando a viabilidade do modelo.

- **Infraestrutura para a computação *Multicluster* em Ambiente Grid [SERRANO, 2006]:**

  O objetivo principal deste trabalho foi a constituição de uma plataforma trans-domínio que permitisse a execução de aplicações desenvolvidas em ambientes de programação por passagem de mensagens, preservando seus ambientes originais numa perspectiva próxima de uma grade computacional.

- ***The Multicluster Model to the integrated use of Multiple Workstation Cluster* [BARRETO; ÁVILA; NAVAUX, 2000]:**

  Expõe e analisa problemas relacionados a integrações de diferentes plataformas em *cluster* e propõe o modelo *MultiCluster* para alcançar a integração desejável utilizando um software desenvolvido pelos pesquisadores tendo como objetivo a união de plataformas *clusters* específicas (Myrinet e SIC) e o fornecimento de uma API uniforme para a programação paralela e distribuída.

Em 2003 o Laboratório Nacional de Computação Científica (LNCC) divulga sua Grade Institucional – GRADE LNCC. Eles apontam em seus estudos que o uso de *clusters* permite a convivência de aplicações que requerem computação fortemente acoplada com aplicações que toleram computação fracamente acoplada bem como facilita a migração e a reformulação das aplicações já existentes. Além disso, a utilização de *clusters* baseia-se na sua difusão devido ao baixo custo, escalabilidade e alternativas de comunicação baratas *intra-clusters*, contando com um único sistema operacional próprio para interligação com o software de grade. Os pesquisadores do LNCC ainda ressaltam que o uso de *clusters* representam uma inovação tecnológica atraente para o setor empresarial o qual demonstra interesse de participar.

Enfim, o projeto GRADE do LNCC tem como objetivo geral a implantação, em nível nacional, de uma grade integrada por uma infraestrutura computacional escalável de *clusters* geograficamente distribuídos (homogêneos e heterogêneos) em diversos centros do país, capaz de permitir o acesso às facilidades computacionais de forma confiável, consistente, ubíqua e de custo acessível.

O que se é importante também apontar é que um ambiente em grade baseado em multicluster ou *clusters* múltiplos permite uma facilidade maior de implementação e implantação, diferente do que ocorre atualmente com o *Globus* [GLOBUS, 2010] que necessita um maior conhecimento do sistema operacional Linux e de sua forma de implantação.

As pesquisas relacionadas a sistemas *MultiCluster*, levam a crer que a implementação de *clusters* múltiplos traz diversos benefícios a um ambiente integrado, tais como: o acesso a recursos computacionais geograficamente distribuídos em *cluster* heterogêneos (a partir deste ponto os *clusters* que incorporarem as grades computacionais serão denominados *clusters* virtuais como propõem Serrano em sua dissertação [SERRANO, 2006]); independência e eficiência com relação a configuração e a portabilidade; maior poder de escalonamento e padronizações de regras garantido a QoS contratada.

2.4. *Web Services* em Ambientes de Grades Computacionais

Partindo do ponto que os *Web Services* têm como principal característica lidar com a interoperabilidade entre sistemas heteregêneos e que as Grades Computacionais são constituídas por recursos pertencentes a diferentes instituições com diferentes domínios administrativos, ferramentas, plataformas, etc., a integração entre *Web services* e Grades trás diversos benefícios.

Os *Web Services* são fortes candidatos para serem utilizados no cenário de Grades Computacionais por diversos motivos: interoperabilidade, recursos de descoberta de serviços providos; geração automática de códigos consumidores; funcionamento em ambientes extremamente heterogêneos e distribuídos; compatibilidade com padrões abertos e possibilidade de serem utilizados no contexto de *Multiclusters*.

Dentro deste contexto, Ian Foster [FOSTER et al, 2002] propôs o OGSA (*Open Grid Services Architecture*) que é uma arquitetura que descreve como os mecanismos de uma grade computacional podem ser implementados usando uma arquitetura SOA (*Service Oriented Architecture*). O objetivo é integrar as tecnologias das grades computacionais com a tecnologia de *web services*. Dentro da arquitetura OGSA foi inserida a idéia de *Grid Services* (Serviços em Grades) onde esses serviços são *Web services* que implementam um determinado conjunto de interfaces padronizadas para resolver os problemas próprios à computação em grade.

Nos últimos anos verificou-se que a arquitetura de grades computacionais e seus padrões vêm adotando um modelo orientado a serviços, utilizando a tecnologia *Web Service*. Tendo em vista que já existe um volume grande de pesquisas sobre este tema, a tecnologia *Web Service* em Grades Computacionais vem se tornando uma tendência é está sendo verificada na prática sua eficácia e seus benefícios.

2.5. Gestão da QoS no Nível de Redes e em Ambientes Distribuídos

O termo QoS (*Quality of Service* – Qualidade de Serviço) indica a capacidade de fornecer um serviço conforme as exigências estabelecidas como, por exemplo, os tempos de resposta de uma requisição ou consulta. Em uma infraestrutura de redes, por exemplo, a QoS funciona como um requisito que exige determinados parâmetros (vazão, atrasos, perdas) dentro de limites bem definidos.

Existem diferentes pontos de vista em relação à aplicabilidade da Qualidade de Serviço em um ambiente computacional, são eles [PINHEIRO, 2004]:

- **Ponto de Vista dos Usuários:** A Qualidade de Serviço é um mecanismo flexível que pode ser modificado no momento em que o usuário considerar adequado.

- **Ponto de Vista dos Programas de Aplicação:** A QoS é estabelecida por meio de um Contrato de Nível de Serviços ou SLA (*Service Level Agreement*) que tem como objetivo especificar detalhadamente os níveis de desempenho que um provedor de serviços deverá manter a disposição do usuário.

- **Ponto de Vista do Gerente ou Administrador:** Neste ponto de vista a QoS é orientada no sentido da utilização de mecanismos, algoritmos e protocolos em benefício de seus usuários, além do suporte às aplicações.

Dentro do conceito de Redes Computacionais a QoS consiste em garantir, por exemplo, a existência de um nível aceitável de perdas de pacotes, ou seja, é garantido através dos componentes e equipamentos utilizados na rede de forma a assegurar a entrega das informações por meio de um mecanismo fim-a-fim, além de atuar na comunicação entre os equipamentos visando o controle dos parâmetros estabelecidos.

O que é importante observar é que essa garantia envolve vários níveis de atuação em diversos tipos de equipamentos e tecnologias, o que quer dizer que os parâmetros não estão localizados em um único componente da rede.

Quando o termo nível de aceitação, atuação ou exigências é colocado, refere-se aos "níveis de serviços" que definem a capacidade relativa aos fluxos de dados de uma rede que fornece serviços a usuários externos ou que façam parte da mesma rede. Os serviços são definidos geralmente em três níveis de QoS:

- **Melhor Esforço (Lack of QoS):** Não fornece nenhuma diferenciação e nenhuma garantia.

- **Serviço Diferenciado (Soft QoS):** Permite definir níveis de prioridade sem fornecer garantia exata.

- **Serviço Garantido:** reserva recursos de rede para certos tipos de fluxos.

No caso das redes de longa distância onde a velocidade de transmissão é dependente da escolha da tecnologia de rede para a garantia da qualidade de serviço, observam-se limitações nas velocidades utilizadas, tipicamente devido aos custos envolvidos na operação da rede além de algumas restrições quanto à disponibilidade tanto da tecnologia quanto da velocidade de transmissão desejada. O resultado é que a garantia de QoS é mais crítica em redes metropolitanas (MAN) e de longa distância (WAN) do que em redes locais (LAN) [PINHEIRO, 2004] .

A Qualidade de Serviço é um aspecto importante para implantação das redes de comunicação tornando-se um aspecto operacional fundamental para o desempenho fim-a-fim das aplicações de redes como voz sobre IP (VoIP), multimídia, entre outras. O que se pode concluir é que a compreensão de seus princípios, parâmetros, mecanismos, algoritmos e protocolos são requisitos para viabilizar uma operação com qualidade de uma aplicação e a obtenção de uma QoS de rede adequada.

Visto da importância da QoS no nível de redes, foram iniciados vários estudos que têm o objetivo de incluir a Gestão da QoS em ambientes distribuídos como *clusters* e grades computacionais. Diversas pesquisas já foram desenvolvidas em relação aos ambientes de *clusters* com o objetivo de criar soluções que possam prover a QoS. Estes estudos tratam de garantias de QoS ligadas ao uso dos recursos de processamento disponíveis e não dos aspectos relacionados ao problema da comunicação. No Capítulo III, é descrito uma solução que aborda o aspecto da QoS considerando a diferenciação de serviços e o balanceamento de carga dentro de um ambiente de *cluster*.

Com o surgimento das grades computacionais, o aumento na utilização das mesmas (seja na área comercial como na acadêmica), a grande variedade de recursos compartilhados e a necessidade de oferecer serviços de melhor qualidade aos usuários, surgiu a necessidade de se incorporar métodos que tivessem como objetivo tratar o problema da QoS de forma a garantir um melhor desempenho do ambiente e o atendimento de forma diferenciada das necessidades dos usuários que utilizam os recursos disponíveis na grade.

Alguns métodos podem ser utilizados a fim de prover QoS dentro de um infraestrutura em grade, entre eles:

- **Controle de Admissão**: O objetivo do controle de admissão é oferecer qualidade de serviço a múltiplas classes, de modo a satisfazer suas restrições de tempo de espera. Ou seja, o controle de admissão tem o objetivo de receber requisições classificadas e realizar o gerenciamento da aceitação pelo servidor, levando em consideração as políticas de atendimento vigentes e as informações da carga de trabalho. Se o servidor estiver sobrecarregado, uma requisição poderá ser rejeitada ou ter suas exigências de qualidade de serviço relaxadas de modo que possa ser aceita em uma classe de prioridade inferior.

– **Balanceamento de Carga**: O objetivo do balanceamento de carga é de equilibrar a carga entre vários servidores físicos, fazendo com que eles pareçam ser um grande servidor para o mundo externo. O principal motivo para a utilização do balanceamento de carga é possibilitar o escalabilidade, alta disponibilidade e previsibilidade.

– **Diferenciação de Serviços:** a diferenciação de serviços permite que os recursos sejam disponibilizados aos usuários da grade com prioridades diferentes para cada tipo de cliente. Permitindo que clientes que estejam dispostos a pagar mais por um serviço de maior qualidade possam utilizar de forma diferenciada os recursos de processamento disponíveis.

2.6. Globus, OurGrid e EuroGrid

Dentre os ambientes em grades computacionais existentes, são frequentemente citados em trabalhos de pesquisas: o *Globus*, o *OurGrid* e o *EuroGrid*.

O *Globus Alliance* [GLOBUS, 2010] é uma colaboração internacional que realiza pesquisas e desenvolvimento de tecnologias fundamentais para grades computacionais. O *Globus Toolkit* inclui serviços de software e bibliotecas de segurança distribuída, gestão de recursos, monitoramento e descoberta, e gerenciamento de dados que facilitam a computação em grade. *Globus* e os protocolos definidos em sua arquitetura tornaram-se um padrão como infraestrutura para computação em grade (Figura 2.2).

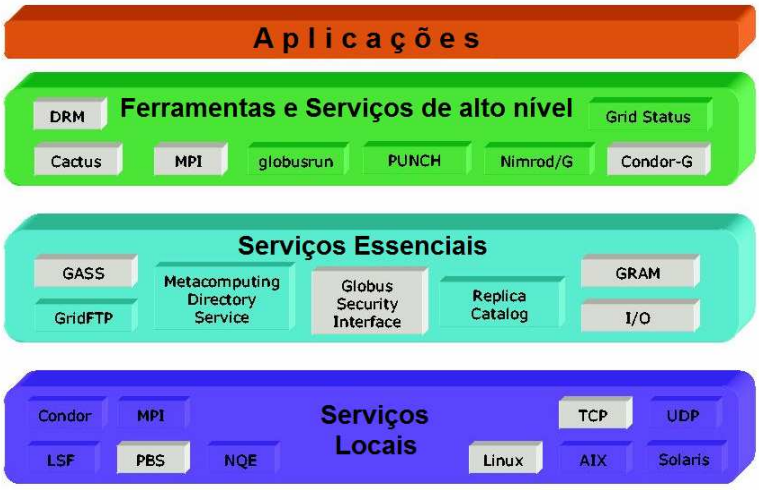

Figura 2.2: *Arquitetura Globus* [GLOBUS, 2010]

O *OurGrid* [OURGRID, 2007] é um projeto peer-to-peer que tem como objetivos principais ser um ambiente com desempenho, simples, escalável e seguro. Foi desenvolvido na Universidade Federal de Campina Grande, para execução de aplicações do tipo *bag-of-tasks* em grades computacionais. O *OurGrid* é dividido em três componentes (Figura 2.3): O *MyGrid* que permite que as aplicações sejam executadas em máquinas da grade; O componente *Peer* que fornece as máquinas da grade para a execução de cada aplicação e o *User Agent* que é o componente que funciona em cada estação da grade capaz de fornecer o acesso a cada máquina.

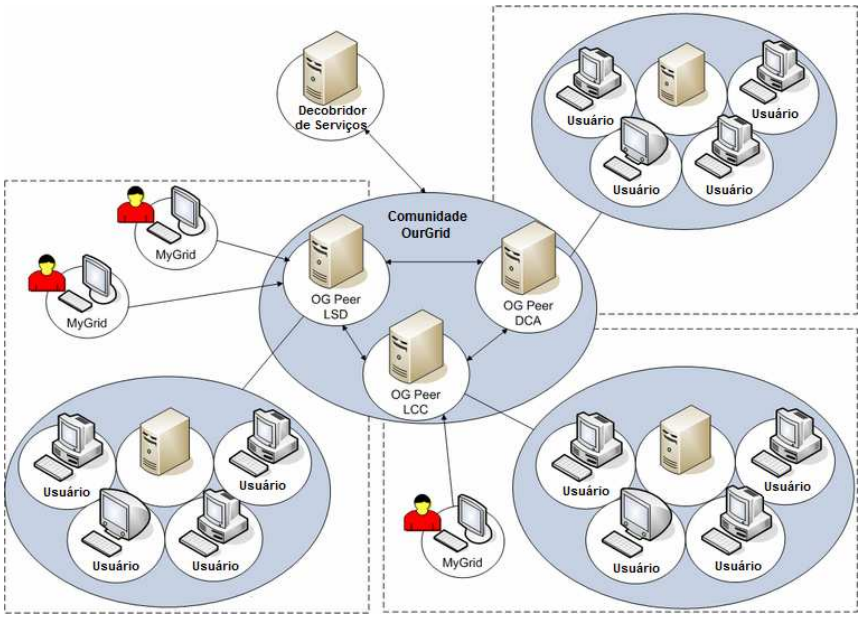

Figura 2.3: *Arquitetura OurGrid* [OURGRID, 2007]

O Projeto *EuroGrid* [EUROGRID, 2004] foi desenvolvido para estabelecer uma rede de grade européia composta pelos principais centros de computação de alta performance entre os diferentes países europeus. O *EuroGrid* é aplicado a domínios de aplicação específicos, tais como: aplicações bio-moleculares, aplicações meteorológicas, aplicações de engenharia e outras aplicações científicas que exigem alta performance computacional.

2.7. Conclusões

Este capítulo apresentou conceitos importantes para o melhor entendimento desta dissertação. Os tópicos abordados neste capítulo (Arquitetura Orientada a Serviços em Grades Computacionais, Grades Computacionais baseadas em Sistemas *MultiClusters*, *Web* Services em

ambientes de Grades Computacionais, Gestão de QoS em nível de Redes e Ambientes Distribuídos) apresentam evoluções ocorridas nos ambientes distribuídos que criaram um contexto que pode ser citado como um dos fatores que motivou o desenvolvimento do presente trabalho.

III

Web Servers – DiffServ Admission Control

WS-DSAC

3.1. Introdução

A arquitetura WS-DSAC (*Web Servers – DiffServ AdmissionControl*) foi desenvolvida voltada para os ambientes em clusters de servidores *Web* utilizando um mecanismo de balanceamento de carga e oferecendo diferentes níveis de QoS através de uma estratégia de diferenciação de serviços que permite o uso eficaz dos recursos de processamento disponíveis.

Este capítulo tem como objetivo apresentar a arquitetura WS-DSAC que serviu como base para o desenvolvimento deste trabalho. Serão apresentados a arquitetura e o mecanismo WS-DSAC com os blocos e os elementos que os compõem assim como os parâmetros e o algoritmo que integram o mecanismo.

3.2. A Arquitetura WS-DSAC

A arquitetura WS-DSAC é composta por três blocos como mostrado na Figura 3.1: bloco de gestão de Qualidade de Serviço – QoS, bloco de monitoramento e no bloco de serviços administrativos. Apesar de cada bloco possuir atividades distintas, foram projetados para funcionar de maneira conjunta visando proporcionar o melhor desempenho possível para a arquitetura.

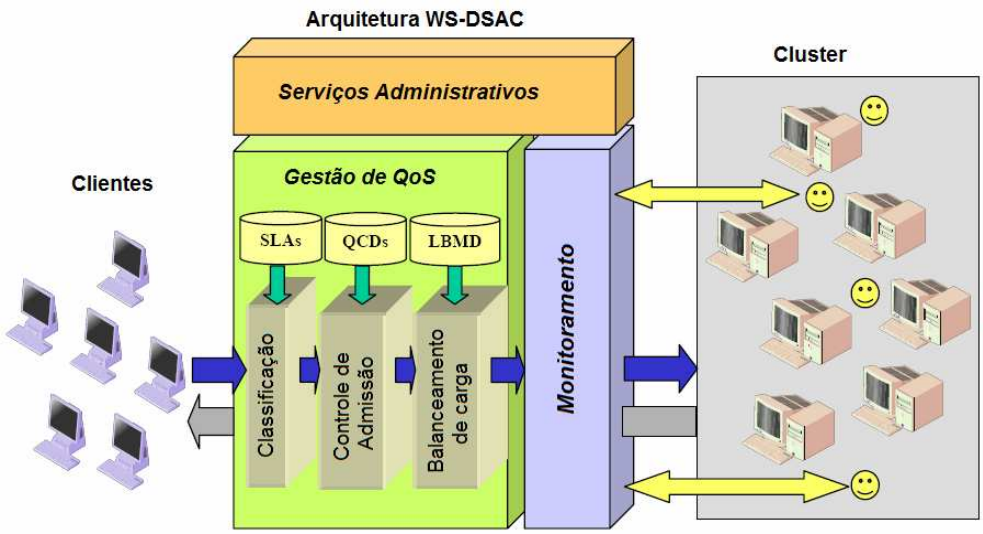

Figura 3.1: ***Arquitetura WS-DSAC*** **[SERRA, 2005]**

3.2.1. Bloco de Gestão de QoS

O bloco de Gestão de QoS possui mecanismos responsáveis pela gestão dos recursos e pela garantia da Qualidade de Serviços. Os elementos que fazem parte deste bloco são:

Mecanismo de classificação: Responsável pela identificação e classificação de requisições e pela política de classificação determinada pelo administrador de recursos. O método de classificação utilizado é flexível e baseado sobre uma política fundamentada em regras que podem ser relacionadas com IPs, Ports, URLs (*Uniform Resource Locator*) ou qualquer outra informação que caracterizem os clientes, fornecedores ou serviços consumidores.

Mecanismo de controle de admissão: O controle de admissão é executado de acordo com os processos de classificação. Uma variedade de políticas é utilizada para decidir quando aceitar ou recusar uma requisição dependendo de sua classificação e das informações das cargas do *cluster*. O objetivo principal deste mecanismo é garantir os parâmetros assegurados para cada classe de QoS existente utilizando simultaneamente, de forma eficaz, os recursos disponíveis.

Mecanismo de balanceamento de cargas: Utilizando informações contidas em uma base LBMS (*Load Balancing Mechanism Specifications*), o mecanismo de balanceamento de carga implementa uma parte do algoritmo distribuído (Mecanismo WS-DSAC) para realizar o balanceamento das cargas nos vários servidores disponíveis dentro da infraestrutura. Este mecanismo é flexível por levar em consideração a distribuição de

cargas gerada por solicitações de entrada para a infraestrutura e por métodos de mensagens/chamadas trocadas dentro do próprio *cluster*.

3.2.2. Bloco de Monitoramento

O bloco realiza o monitoramento e a estimativa de cargas do sistema. Ele fornece as informações necessárias ao Bloco de Gestão de QoS para a tomada de decisões ligadas ao controle de admissão e para o balanceamento de cargas. Ele coloca também à disposição as informações sobre o comportamento do sistema para a tomada de decisões relativas à realocação dos recursos.

O bloco de monitoramento tem um papel importante dentro do processo que o torna responsável pela medição, montagem, tratamento e disponibilização das informações necessárias para o equilíbrio do sistema. Para a medição de cargas em nível de servidores é utilizada uma solução baseada em um "coeficiente de reatividade" [SERRA; BARROSO; BOUDY, 2004]. No tópico 3.3. "Visão Geral da Plataforma" o "coeficiente de reatividade" será abordado com mais detalhes.

3.2.3. O Bloco de Serviços Administrativos

O bloco de Serviços Administrativos serve de interface entre o administrador de recursos e o sistema. Utilizando as funcionalidades desses serviços, o administrador pode gerir os recursos. Entre outras atividades, ele pode:

- Definir e atualizar as classes de QoS, especificando seus respectivos parâmetros;
- Gerir informações relacionadas a contratos de QoS (SLAs) concedidos entre clientes e prestadores dos serviços;
- Dispor de estatísticas sobre o comportamento do sistema em relação à gestão de recursos e à QoS oferecida;
- Executar as atividades administrativas relacionadas com a conexão on-line ou desativação de aplicações ligadas ao *cluster*;

- Gerir as informações relativas à cobrança pela utilização dos recursos tanto para clientes quanto para os fornecedores dos serviços.

Neste bloco, existem, ainda, serviços disponíveis para os clientes e os fornecedores dos serviços, permitindo:

- Se registrar como cliente ou fornecedor;
- Colocar *on-line* as aplicações *Web* de forma automática;
- Consultar as informações relativas à utilização dos recursos e à QoS fornecida.

3.3. Visão Geral da Arquitetura

A Arquitetura WS-DSAC (Figura 3.2) é composta por um conjunto de elementos básicos: *Class Switch*, *Cluster Gateways* e *Web Server Nodes*. O *Class Switch* é responsável pela classificação e pelo controle de admissão de novas requisições de clientes. Ele recebe requisições HTTP (*HyperText Transfer Protocol*), identifica a classe do serviço e, utilizando o mecanismo 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*.

Os serviços são instalados em um certo número de servidores *Web*. Estes serviços podem ser compostos por serviços *Web* e objetos distribuídos. Cada Servidor *Web* possui um agente monitor que monitora a sua carga. O agente monitor usa um “Coeficiente de Reatividade” [SERRA; BARROSO; BOUDY, 2004] que fornece uma idéia da tendência da carga do servidor. Esta técnica mede o tempo que uma tarefa espera para ter acesso ao tempo de CPU. O tempo de espera depende diretamente da carga do servidor.

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

usa a mesma informação de carga fornecida pelo CRM para manter o balanceamento de cargas dentro do domínio da classe[1].

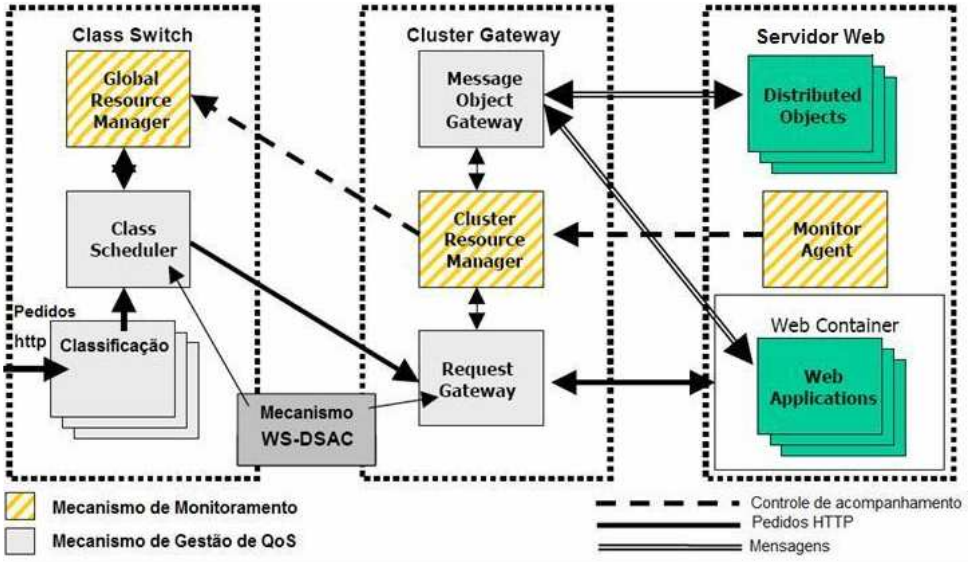

Figura 3.2: ***Componentes do mecanismo WS-DASC*** **[SERRA, 2005]**

A plataforma oferece diferentes níveis de QoS, um nível diferente para cada classe de serviço. 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*. No *Class Switch* um componente denominado GRM (*Global Resource Manager*) solicita e mantém informações de carga de cada *cluster*. Quando uma requisição HTTP é recebida, o *Class Switch* identifica a classe da requisição e redireciona a requisição para o *cluster* apropriado utilizando o mecanismo WS-DSAC. Caso a QoS especificada para classe não possa ser fornecida, ele devolve uma mensagem recusando a requisição.

3.4. O Mecanismo WS-DSAC

O Mecanismo WS-DSAC tem como propósito principal oferecer uma solução flexível e pouco intrusiva fornecendo Qualidade de Serviço para aplicações *Web* utilizando de forma otimizada os recursos de processamento disponíveis. Ele é um algoritmo distribuído onde as partes são executadas pelos mecanismos de gestão concebidos para a plataforma, mais especificamente pelos componentes *Class Scheduler* e «*httt request Gateway*».

[1] Domínios de classes são agrupamentos de servidores *Web* dedicados a atender prioritariamente requisições pertencentes a uma classe de QoS específica.

O mecanismo tem três objetivos principais:

- Realizar um balanceamento de cargas em dois níveis, entre os servidores de um *cluster* e entre um conjunto de domínios de classes;

- Permitir a diferenciação de serviços em relação ao uso dos recursos de processamento disponíveis em clusters de servidores *Web*, garantindo assim diferentes tempos de resposta para diferentes classes de serviço;

- Utilizar de forma eficaz recursos de processamento disponíveis realocando de forma dinâmica os recursos entre classes diferentes de serviços enquanto estes não estão em níveis críticos de utilização de forma que, serviços de classes menos prioritárias não sejam sacrificados por serviços mais prioritários enquanto existirem recursos disponíveis.

O algoritmo que compõe o mecanismo WS-DSAC foi concebido de maneira a respeitar as características que abordam a reflexão relativa à concepção, a modelagem e a implementação de uma solução que forneça a QoS para aplicações *Web*. Essas características são [SERRA, 2005]:

Aplicação de Princípios para o Fornecimento de QoS: Esses princípios incluem a classificação de serviços, controle de admissão e uso eficiente de recursos.

Equilíbrio entre Inteligência e Desempenho: O equilíbrio entre "inteligência" e desempenho dos mecanismos utilizados é uma questão complexa. A solução foi trabalhada em dois níveis de inteligência: inteligência em um nível mais básico, para a programação e o roteamento inteligente, e em um nível mais sofisticado de gestão de recursos. Na primeira, a estratégia é usar técnicas simples de tratamento de baixo custo computacional para dar um pouco de inteligência aos mecanismos que afetam diretamente o desempenho e, em seguida, usar técnicas mais sofisticadas, de aprender com o comportamento desses mecanismos e de agir proativamente em seus parâmetros.

Modularidade, Escalabilidade e Portabilidade: A modularidade consiste em um software de elementos que compõem a solução elaborada de forma independente, como componentes. Eles funcionam de forma centralizada ou distribuída para permitir maior flexibilidade na implantação de infraestrutura física. Para isso, foi utilizado orientação a

objetos. A escalabilidade destina-se a permitir o uso escalável de um grande número de servidores. No entanto, também permite a criação de plataformas mais modestas, até mesmo com dois servidores no total. A portabilidade leva em consideração diversas tecnologias de software presentes ao desenvolver uma aplicação *Web*. Ela também pode ser estendida para acomodar a utilização de novas tecnologias e softwares além de funcionar em sistemas operacionais diferentes.

Diferenciação de QoS: Um conceito importante relacionado à QoS e aplicações *Web* é o contrato de QoS ou contrato de SLA (*Service Level Agreement*). O serviço dispõe de um conjunto de características expressas em termos quantitativos ou estatísticos de forma a esclarecer as garantias oferecidas pelo fornecedor de serviços. Ou seja, um SLA é um contrato entre o prestador de um serviço *Web* e o cliente onde são especificados os termos mensuráveis de QoS a serem fornecidos.

Uso Eficiente de Recursos: Os mecanismos procuram utilizar os recursos da forma mais eficiente possível, mas ao mesmo tempo devem garantir a QoS prometida para cada classe de QoS. Para isso, os recursos de processamento disponíveis inicialmente são particionados de maneira lógica em “domínios de classes” onde um conjunto de parâmetros que define a QoS oferecida é especificada a cada um deles.

Unidade de Medida Flexível para QoS: a QoS oferecida ao usuário de um sistema baseado na *Web* é o resultado da QoS fornecida pela rede e pelo servidor *Web*. Nas redes, a medição da QoS é feita geralmente através de parâmetros como a taxa de perda, atraso, variação de retardo, etc. No lado servidor a QoS pode ser medida através do tempo de processamento da requisição.

Alocação Dinâmica de Recursos: Mecanismos de partição de recursos são utilizados para permitir a utilização diferenciada de recursos pelas requisições dos clientes. Estes mecanismos podem utilizar tanto um particionamento estático como dinâmico, estabelecido pelo *Web Switch*[2], ou uma prioridade de agendamento, estabelecida pelos servidores.

[2] *Web Switch*: faz o roteamento do tráfego de servidores *Web* apropriado com base na URL ou endereço IP. O *Web Switch* também é conhecido como um "*URL Switch*".

Menor Intrusão Possível: As soluções introduzidas em um middleware ou um sistema operacional são por vezes bem sucedidas, mas elas têm um custo de instalação e manutenção elevadas para os fornecedores de serviço porque eles são forçados a usar softwares adaptados ou sistemas operacionais proprietários.

Formalmente, o mecanismo leva em consideração um conjunto de parâmetros: a média de cargas dos servidores *Web* registrados nos domínios de classes, a carga média estimada em um período de tempo dos servidores *Web* da classe do *cluster* e o valor da carga estimada do domínio de classe do *cluster* com menor carga por período.

O mecanismo funciona da seguinte forma: a cada período de tempo o *Cluster Resource Menagers* - CRM interroga todos os servidores sob o seu domínio a fim de obter as cargas atuais e as cargas estimadas para o próximo período. Em seguida, o CRM estima a carga da classe inteira para sua "classe nativa[3]". Baseado nessas estimativas e pelos parâmetros de cada classe de QoS é determinado o "modo de trabalho" (compartilhado, exclusivo e saturado) de um "domínio de classe" para os próximos períodos de tempo.

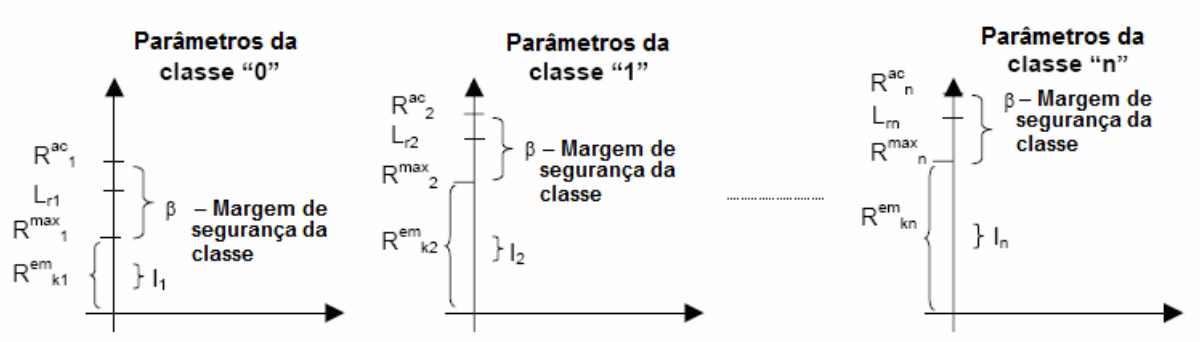

Figura 3.3: ***Parâmetros de Configuração do Mecanismo WS-DSAC*** **[SERRA, 2005]**

Quando uma requisição chega em um determinado intervalo de tempo, o mecanismo WS-DSAC é aplicado. Antes, o mecanismo de classificação determina a classe de QoS da requisição. Logo que é determinada a classe de QoS para a requisição e qual "domínio de classe" do *cluster* está menos carregado, o algoritmo faz a tomada de decisão por meio de um evento condicional, elegendo o servidor menos carregado dentro do "domínio de classe" para o processamento.

[3] Classe nativa: classe de QoS a qual o servidor estar alocado para atender prioritariamente suas requisições.

Na Figura 3.3 são ilustrados os parâmetros de configuração do mecanismo WS-DSAC. No capítulo IV no tópico que apresenta o mecanismo G-DSAC, estes parâmetros são descritos e a forma de cálculo do “modo de trabalho” dos “domínios de classe” é apresentada.

3.5. Avaliação da Plataforma WS-DSAC

Um protótipo da arquitetura foi implementado por Serra [SERRA, 2005] sobre uma infraestrutura real de teste com o intuito de ilustrar a eficácia do mecanismo WS-DSAC.

Os experimentos mostraram que a técnica utilizada (coeficiente de reatividade) para a medição da carga exprime bem as variações de mudanças de carga dos servidores e possui uma forte correlação com o tempo de respostas aos usuários [SERRA, 2005].

A fim de avaliar se o mecanismo WS-DSAC era capaz de atingir os objetivos propostos, a plataforma foi colocada em diferentes situações de tráfego e foi constatado que o mecanismo é capaz de alocar os recursos de uma maneira equitativa durante os momentos de baixa carga através de uma realocação dinâmica de recursos e que é, também, suficientemente responsável por garantir os contratos de Gestão de QoS (SLAs) estabelecidos mesmo dentro de momentos de sobrecarga.

Por fim, foram realizados experimentos com o intuito de analisar o *overhead* imposto pela solução dentro do caso específico do protótipo implementado.

A conclusão para os experimentos foi que o mecanismo atingiu com sucesso os objetivos propostos para a arquitetura WS-DSAC.

3.6. Conclusões

Este capítulo apresentou uma visão geral da arquitetura WS-DSAC, que foi desenvolvida com foco nos ambientes em clusters de servidores *Web*, e dos blocos que a compõem: o bloco de gestão de Qualidade de Serviço, bloco de monitoramento e o bloco de serviços administrativos. Uma visão sobre o WS-DSAC foi apontada descrevendo as funcionalidades dos elementos básicos que fazem parte da plataforma: *Class Switch*, *Cluster Gateways* e *Web Server Nodes*. A

descrição do mecanismo WS-DSAC, seus objetivos, características e seu funcionamento foram brevemente apresentados.

Em relação aos experimentos realizados na avaliação da arquitetura WS-DSAC [Serra, 2005], ficou constatado que os objetivos propostos pela arquitetura foram atingidos demonstrando que a solução oferece diversos benefícios dentro de um ambiente em *cluster* em relação ao balanceamento de carga, a gestão dos diferentes níveis de QoS e ao uso eficaz dos recursos de processamento disponíveis.

Pôde-se observar que os benefícios oferecidos pela arquitetura WS-DSAC são voltados para os ambientes em *clusters* de servidores *Web*. Neste trabalho, esta arquitetura serviu como base para o desenvolvimento de uma nova arquitetura que leva os benefícios por ela oferecidos para um ambiente de grades computacionais orientado a serviços.

IV

Grid – DiffServ Admission Control

G-DSAC

4.1. Introdução

A arquitetura G-DSAC (*Grid – DiffServ Admission Control*) foi concebida com foco nas arquiteturas em grade orientada a serviços. A Arquitetura G-DSAC é uma extensão da arquitetura WS-DSAC. Ela compreende a concepção de uma solução voltada para grades computacionais [NASSIF; NOGUEIRA, 2007] que é capaz de garantir os SLAs (*Service Level Agreements*) estabelecidos com os consumidores de serviços utilizando de forma otimizada os recursos de processamento disponibilizados na grade. Permite ainda a diferenciação de serviços no que diz respeito aos tempos de resposta oferecidos aos "clientes[4]", usuários finais e serviços consumidores.

Neste capítulo são abordados aspectos relacionados à nova plataforma proposta, iniciando pela declaração do problema ou motivação, seguido da descrição da arquitetura e de seus elementos, da visão geral da plataforma, da descrição do mecanismo G-DSAC e dos trabalhos relacionados.

[4] No contexto deste trabalho, o termo cliente faz referência a um código de um determinado programa que faz a interface entre o usuário final e o ambiente da grade.

4.2. Contexto

Um ponto crítico a ser tratado dentro das arquiteturas em grade orientadas a serviços é a questão da gestão da QoS (*Quality of Service*), no que diz respeito ao uso dos recursos de processamento, oferecida aos usuários da grade. Levando em consideração que o ponto de sobrecarga (ponto de estrangulamento) vem se localizando cada vez mais nos recursos de processamento, duas possíveis soluções largamente utilizadas para lidar com este problema e aumentar a QoS do usuário final são: a distribuição equilibrada da carga imposta pelos serviços dentro da grade e o uso de mecanismos de controle de admissão para evitar a sobrecarga ou possível "crash" de servidores [CASTRO; DWYER; RUMSEWICZ, 1999] [CHERKASOVA; PHAAL, 2002] [ABDELZAHER; SHIN; BHATTI, 2002].

Por outro lado, usuários com expectativas diferentes em relação à QoS desejada, mesmo podendo pagar mais caro pelos serviços utilizados, são forçados a compartilhar de forma igual os recursos de processamento disponibilizados para o provimento de um serviço na WWW. No nível de redes baseadas no protocolo IP, mecanismos como o DiffServ [CARPENTER; NICHOLS, 2002] ou MPLS [ROSEN et al., 2001], permitem a diferenciação de serviços agrupando-os em classes e garantindo a cada classe certos parâmetros de QoS (perdas, demora, variação da demora). No nível de aplicação, para permitir que fornecedores de serviços possam diferenciar a QoS oferecida aos seus clientes, é necessária a utilização de mecanismos que permitam o uso diferenciado dos recursos de processamento disponíveis.

Um exemplo deste tipo de mecanismo está implementado na arquitetura WS-DSAC [SERRA, 2005] (Capítulo III). O mecanismo realiza o controle de admissão e o balanceamento de cargas dos servidores *Web* além de permitir a diferenciação de serviços através do agrupamento de requisições em classes diferentes de serviços. Um parâmetro de QoS denominado "Coeficiente de Reatividade" [SERRA; BARROSO; BOUDY, 2004] (diretamente relacionado ao tempo de resposta do cliente) é associado a cada classe de serviços pré-estabelecida. O mecanismo 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.

No entanto, quando a arquitetura WS-DSAC foi concebida, levou em consideração ambientes baseados em clusters de servidores *Web*, ou seja, a arquitetura WS-DSAC não trata

conceitos importantes ligados a orientação a serviços e a computação em grade, não se adequando aos novos paradigmas da computação distribuída, notadamente as Arquiteturas em Grade Orientadas a Serviços.

4.3. Estendendo a arquitetura WS-DSAC

Como a arquitetura WS-DSAC não leva em consideração o uso de *Web Services*, uma série de ajustes (Figura 4.1) foi realizada para que na arquitetura G-DSAC este novo conceito pudesse ser aplicado em um ambiente em grade computacional (ver Capítulo V).

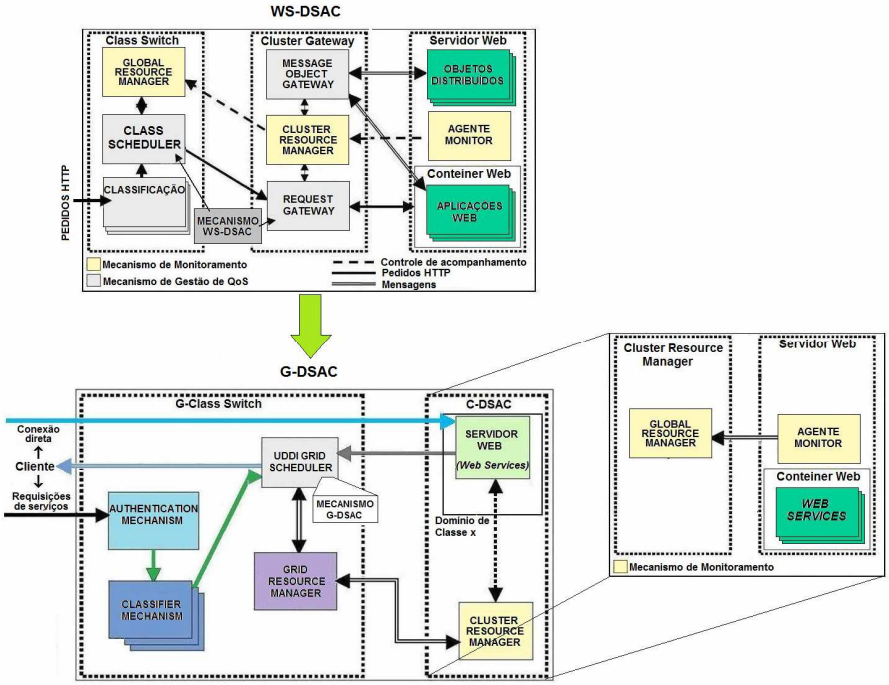

Figura 4.1: ***Transição da arquitetura WS-DSAC para a arquitetura G-DSAC***

Na Figura 4.1 são apresentados os elementos da arquitetura WS-DSAC original (ver Capítulo III) e em seguida os elementos da arquitetura G-DSAC. Os elementos da nova Arquitetura serão apresentados a seguir.

4.4. A Arquitetura G-DSAC

Neste trabalho é proposta uma nova Arquitetura denominada G-DSAC (*Grid – DiffServ Admission Control*) (Figura 4.2) que tomou como base a Arquitetura WS-DSAC. A arquitetura

inicial foi revista e transformada em uma nova arquitetura que leva em consideração aspectos relacionados à orientação a serviços e a computação em grade.

Na nova Arquitetura é proposta a introdução de um bloco que implementa um conjunto de funcionalidades em uma plataforma em grade computacional formada por *Multclusters* (Figura 4.3). Este bloco está responsável pela: publicação e localização de serviços, autenticação e classificação de requisições e escalonamento das mesmas dentro da grade.

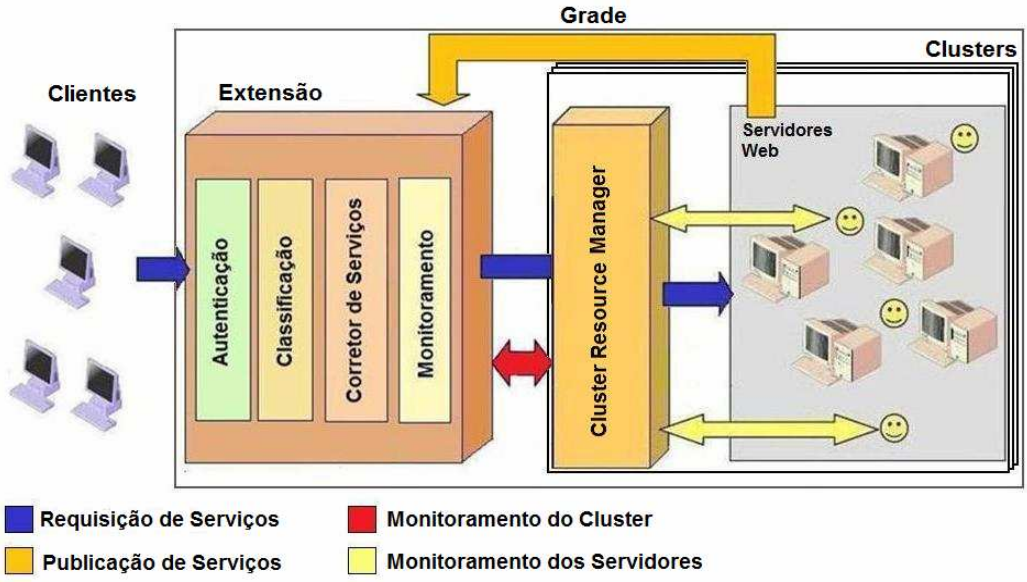

Figura 4.2: ***Arquitetura G-DSAC***

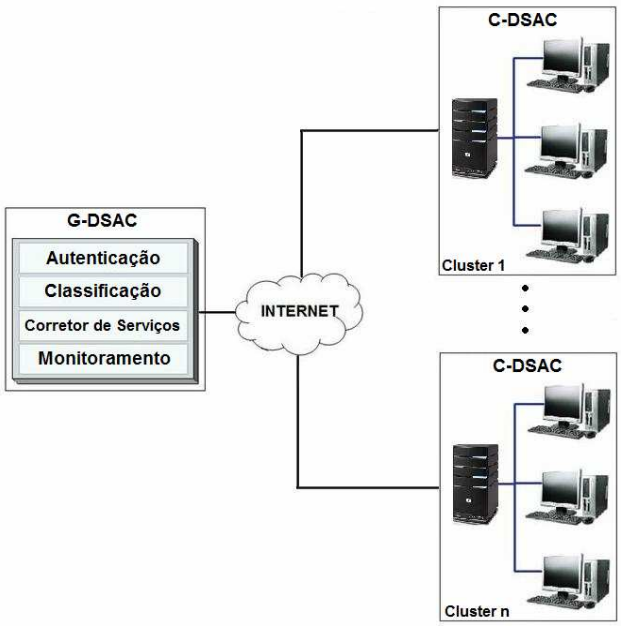

Figura 4.3: ***Visão Geral da Plataforma G-DSAC***

A Arquitetura G-DSAC contempla aspectos de uma arquitetura em grade orientada a serviços tomando como base o mecanismo WS-DSAC. Ela permite: a descoberta e publicação de serviços *Web*, o controle de admissão, o balanceamento de cargas e a oferta de diferentes níveis de QoS. A seguir, os elementos que compõem a nova arquitetura são descritos (Figura 4.4).

4.4.1. O Elemento *Grid Class Switch*

A extensão proposta introduz um novo elemento na Arquitetura denominado *Grid Class Switch* ou *G-Class Switch* que se difere do *Class Switch* da arquitetura original WS-DSAC (ver Capítulo III – Fig. 3.2), pois no lugar de distribuir as requisições dentro de um *cluster*, passou a distribuir dentro de uma grade. O *G-Class Switch* é composto por um conjunto de componentes básicos (Figura 4.4):

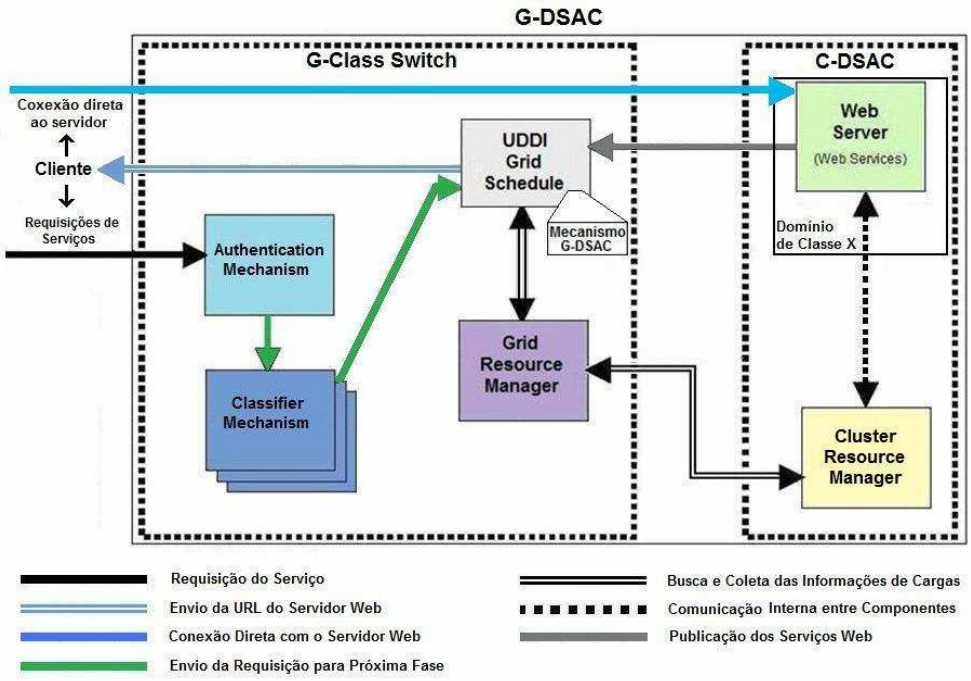

Figura 4.4: ***Componentes da Extensão***

Authentication Mechanism: Responsável em identificar a origem da requisição e autenticá-la;

Classifier Mechanism: Responsável pela classificação da requisição de acordo com a política de classificação selecionada pelo administrador da grade;

Grid Resource Manager: Responsável pelo monitoramento das cargas dos diversos elementos que compõem a grade;

UDDI Grid Scheduler: Elemento responsável pela publicação, e descoberta de serviços dentro da grade levando em consideração aspectos ligados à QoS (balanceamento de cargas e diferenciação de serviços).

Além dos componentes básicos faz parte da arquitetura um algoritmo distribuído denominado “Mecanismo G-DSAC” que realiza cálculos por meio de parâmetros de maneira semelhante ao “Mecanismo WS-DSAC”, indicando qual o *cluster* menos carregado e em seguida qual o servidor *Web* que possui o serviço e que melhor atenderá a requisição levando em consideração a QoS desejada. O mecanismo foi projetado visando utilizar de forma eficaz os recursos de processamento disponíveis e promover a equidade no uso dos mesmos (Tópico 4.4.2).

Como já descrito anteriormente, os serviços são instalados num certo número de servidores *Web* que estão localizados em *clusters* distribuídos geograficamente. Estes serviços são compostos por *Web Services*. Cada Servidor *Web* possui um agente monitor que monitora a sua carga. Os dados dos servidores *Web* (IP, domínio de classe e carga) são enviados para o *Cluster Resource Manager* (CRM) que armazena os dados de carga e mantém esses dados à disposição do *Grid Resource Manager*. Já o *Grid Resource Manager* (GdRM), que se encontra na extensão (na grade), monitora as cargas dos diversos *clusters* que compõem a grade e mantém essas informações à disposição do *UDDI Grid Scheduler*.

Os tópicos seguintes ilustram, de forma geral, o funcionamento da arquitetura proposta.

Autenticação (*Authentication Mechanism*) e Classificação (*Classifier Mechanism*)

Ao receber uma requisição, o elemento *Authentication Mechanism* faz a autenticação da mesma, de acordo com a política de classificação adotada pelo administrador da grade, e com base nas informações contidas na mesma (Figura 4.5). As requisições podem ser autenticadas por diferentes critérios: pela identificação do cliente, pelo tipo de serviço, pelo IP de origem, pela rede de origem, etc. Logo que a requisição é autenticada, ela é enviada para o processo de Classificação. Caso o processo de

autenticação falhe e a requisição não seja identificada, a grade retorna um erro para o solicitante (consumidor do serviço ou cliente).

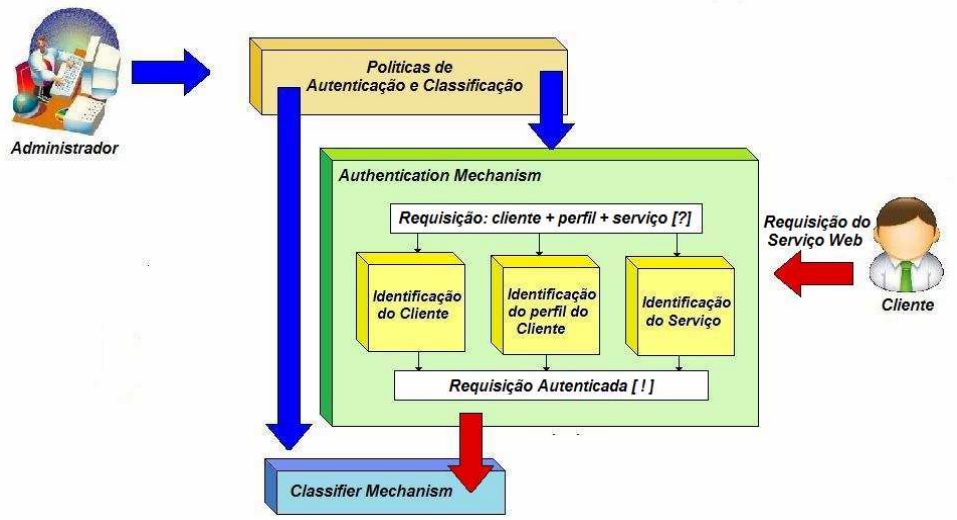

Figura 4.5: ***Funcionamento do Mecanismo de Autenticação***

Como ilustrado na Figura 4.6, finalizada a fase de autenticação, a próxima etapa compreende a classificação da requisição que determina o nível de QoS que deve ser atribuída de acordo com a política adotada e com o contrato de serviço estabelecido.

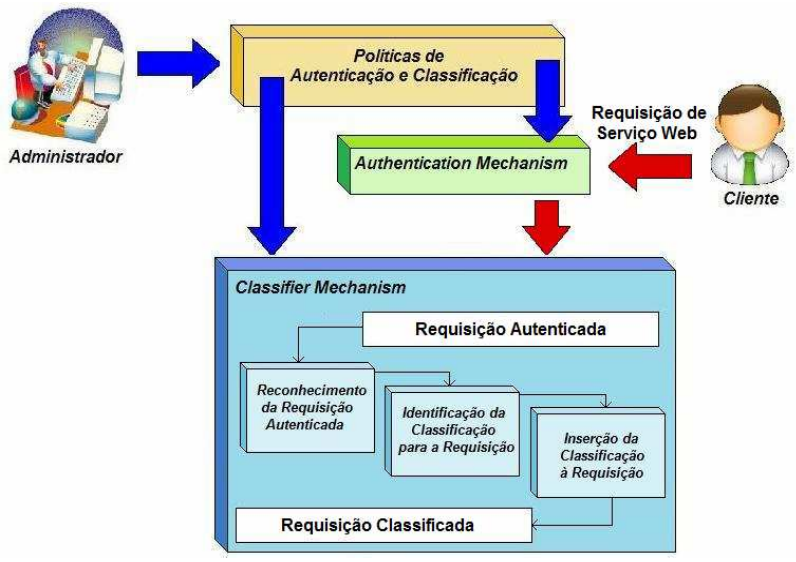

Figura 4.6: ***Funcionamento do Mecanismo de Classificação***

A requisição autenticada é recebida pelo mecanismo de classificação – *Classifier Mechanism* que atribui uma classe de QoS à requisição (Figura 4.6). Após este processo, a requisição classificada irá possuir as informações relevantes que serão utilizadas pelo

UDDI Grid Scheduler (repositório dos serviços publicados) para o processo de descoberta e escolha do serviço na grade.

Publicação e Descoberta de Serviços na Grade (*UDDI Grid Scheduler*)

O componente *UDDI Grid Scheduler* funciona como um catálogo de serviços que contém informações sobre os serviços publicados na grade. É através de elementos (Figura 4.7) que integram este componente que é possível saber: a localização, o modo de trabalho (compartilhado, exclusivo e saturado) e a QoS ofertada, entre outras informações.

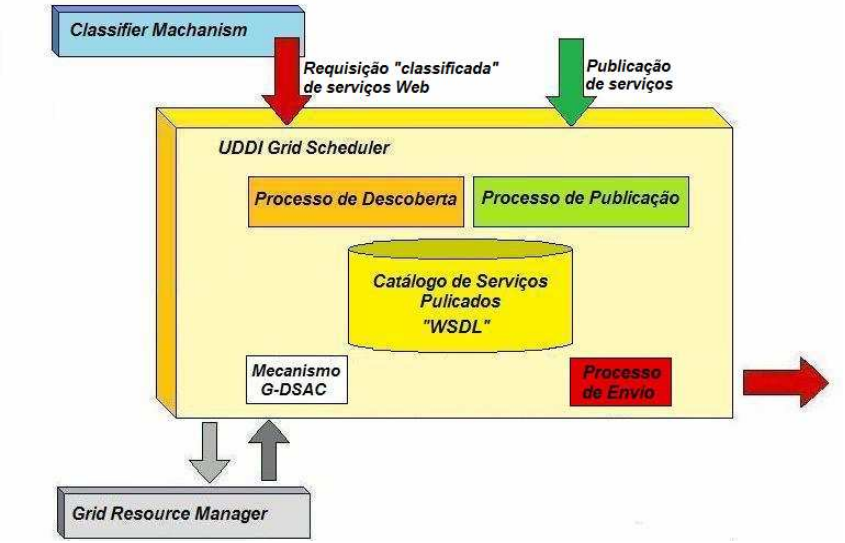

Figura 4.7: ***Elementos do UDDI Grid Scheduler***

Para um serviço tornar-se disponível na grade é necessário realizar o processo de publicação que consiste de um registro de suas informações. Este processo se inicia quando o responsável pelo fornecimento de um serviço na grade faz o acesso ao portal de serviço *UDDI Grid Scheduler* por meio de uma URL (*Uniform Resource Locator*) (Figura 4.8). Os dados relativos ao serviço tais como: tipo do negócio, categoria, organização hierárquica são registrados e o arquivo WSDL[5] criado é publicado. Como outros servidores UDDI já existentes, O *UDDI Grid Scheduler* também utiliza a Linguagem de Descrição de *Web Service* (*Web Service Description Language* - WSDL) [W3C, 2002]. É através dos dados constantes no documento WSDL que é possível saber quais operações o serviço possui, quais os tipos de entrada e saída, além da localização

[5] WSDL – Linguagem baseada em XML utilizada para descrever *Web Services*. Trata-se de um documento escrito em XML que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis.

física e o tipo do protocolo utilizado para a descoberta. Após a conclusão da publicação do serviço, o mesmo já poderá ser requisitado pelos clientes da grade.

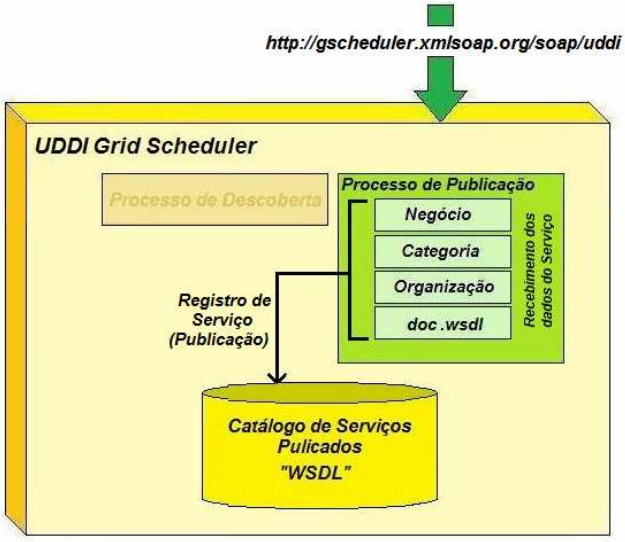

Figura 4.8: ***Processo de Publicação de Serviços***

O processo de Descoberta (Figura 4.9) é a entrada de uma requisição realizada pelo cliente da grade após ter passado pelos procedimentos de autenticação e classificação. O serviço é descoberto através das informações contidas no documento WSDL publicado pelo fornecedor que compõe a grade, e a URL do servidor *Web* menos carregado é enviada para o cliente para que seja realizada a conexão direta obedecendo aos acordos de QoS (SLAs) pré-definidos. No momento da descoberta poderá acontecer do serviço requisitado estar hospedado em apenas um servidor *Web* ou estar replicado em outros servidores *Web* de um mesmo *cluster* ou em *clusters* diferentes.

As informações relevantes para a seleção do servidor *Web* que melhor atenderá a requisição são: a classe de QoS da requisição, a localização do serviço e as informações de carga do servidor.

O processo de descoberta obedece ao seguinte curso: a requisição classificada é enviada para o *UDDI Grid Scheduler* que executa a busca no repositório onde os serviços foram publicados. Após a localização do serviço e do(s) ponto(s) de hospedagem, os dados da requisição são enviados ao mecanismo G-DSAC e os dados da(s) localização(ões) são enviados para o componente *Grid Resource Manager*. O GdRM faz uma varredura em seu “arquivo de monitoramento”, seleciona as informações de cargas para o(s) local(is) descoberto(s) e envia esta informação para o mecanismo G-DSAC.

Como será explanado no tópico 4.4.2, o mecanismo recebe os parâmetros e executa o algoritmo que indica qual o servidor *Web* mais apropriado para atender a requisição enviada pelo cliente de acordo com o seu SLA.

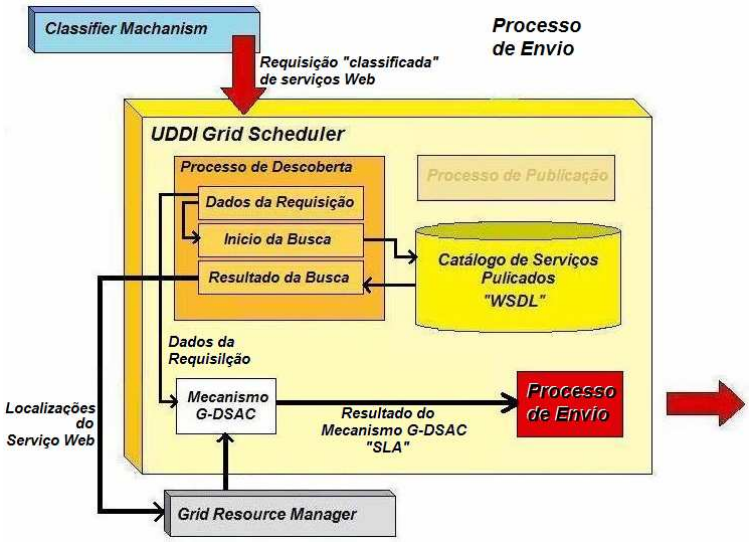

Figura 4.9: ***Processo de Descoberta de Serviços***

Gerenciamento dos Recursos de Processamento - GRM

Como mostrado na Figura 4.10 o elemento *Grid Resource Manager* (GRM) possui duas funções distintas. A primeira é a realização do monitoramento das cargas dos clusters pertencentes à Grade Computacional que consiste em coletar os dados relevantes tais como: médias das cargas, cargas atuais e cargas estimadas, em intervalos de tempo específicos e disponibilizá-los para a realização do processo de descoberta.

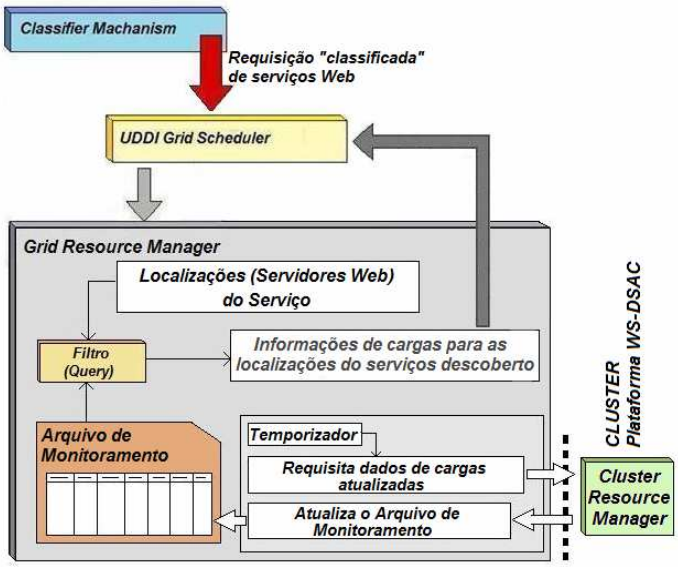

Figura 4.10: ***Funcionamento do componente Grid Resource Manager***

A segunda função consiste em disponibilizar para o Mecanismo G-DSAC, que compõe o elemento *UDDI Grid Scheduler,* as informações relativas a localização e a carga dos servidores *Web* onde o serviço se encontra disponível. Com base nessas informações o mecanismo G-DSAC decidirá qual servidor melhor atenderá a requisição de acordo com o resultado do algoritmo descrito no tópico 4.4.2.

No Capítulo V será apresentado um detalhamento do funcionamento do componente *Grid Resource Manager,* bem como as tecnologias utilizadas para sua implantação.

4.4.2. O Mecanismo G-DSAC

Assim como o "Mecanismo WS-DSAC", o "Mecanismo G-DSAC" tem como propósito principal oferecer uma solução flexível e pouco intrusiva provendo Qualidade de Serviços através do uso eficaz dos recursos de processamento disponíveis. Na arquitetura, o mecanismo G-DSAC é implementado no componente *UDDI Grid Scheduler* onde é executado.

O mecanismo tem três objetivos principais:

- Prover a diferenciação de serviços no nível de grade computacional no processamento das requisições;
- Promover balanceamento de cargas no nível de grade, *cluster* e domínios de classe;
- Utilizar de forma otimizada os recursos de processamento disponíveis na grade.

O algoritmo G-DSAC foi adaptado a partir do WS-DSAC para respeitar as características relacionadas à concepção, à modelagem e à implementação de uma solução que forneça a QoS para aplicações *Web* em uma grade computacional orientada a serviços.

O Mecanismo G-DSAC é definido formalmente pelos seguintes parâmetros:

T1, T2, …, Tn,… – Sequência de intervalos de tempo utilizados para a tomada de decisão relativa à admissão ou rejeição das requisições durante o próximo período de tempo. A duração deste intervalo é especificada pelo parâmetro «ac - intervalo».

N_c – Número de clusters diferentes definidos dentro da grade ($N_c \geq 1$);

N_d – Número de domínios de classes de QoS diferentes definidos dentro da plataforma ($N_d \geq 1$);

N_{sij}– Número de servidores *Web* «s» alocados nos domínios de classes de QoS «i» de um determinado *cluster* «j», ($j = 1..N_c$) e ($i = 1..N_d$);

R_i^{ac} – Valor limite que o «coeficiente de reatividade» do domínio de classe «i» pode atigir ($i = 1..N_d$);

R_{ki}^{em} – Valor limite do «coeficiente de reatividade», recalculado de forma dinâmica a cada período de tempo T_k, atingido pela classe «i» durante o $k^{éma}$ período de tempo, a partir do qual a grade passa a trabalhar em «modo exclusivo»;

r_{ksij} – Carga (média do «coeficiente de reatividade») do servidor *Web* «s» dentro da classe de QoS «i», dentro de um *cluster* «j», medida durante o $k^{éma}$ período de tempo, ($j = 1..N_c$), ($i = 1..N_d$);

r_{kij} – Carga média dos servidores *Web*, registrados dentro de um domínio de classe de QoS «i», pertencente a um *cluster* «j», medido durante o $k^{éma}$ período de tempo;

$$r_{kij} = \left(\frac{\sum_{i=1;j=1}^{N_d;N_c} r_{ksij}}{N_{sij}} \right)$$

ρ_{ksij} – Carga estimada para o período de tempo (k+1) do servidor *Web* «s», pertencente à grade de classe de QoS «i», dentro de um *cluster* «j», calculada durante o $k^{éma}$ período de tempo a partir da função de controle de admissão «f_{ac}» após o intervalo T_k e antes que o

intervalo $T_{(k+1)}$ inicie, ou seja: $\rho_{ksij} = f_{ac}(k+1)$. A função de controle de admissão é dada por:

$$1)\ f_{ac}(1) = R_i^{ac};$$

$$2)\ f_{ac}(k+1) = (1-\alpha) * f_{ac}(k) + \alpha * r_{ksij};$$

Em que o α é um coeficiente entre 0 e 1, denominado coeficiente de ponderação do controle de admissão. Esse coeficiente permite ao mecanismo combinar duas propriedades diferentes: responsabilidade e estabilidade. Quando o valor de α é utilizado mais próximo de 1, significa que a última leitura de carga terá peso maior na estimativa da próxima carga. Quando o valor de α está próximo de 0, a estimativa da próxima carga atribui maior peso ao histórico das cargas considerando menos a última carga mensurada.

r_{kj} – Carga média do *cluster* «j» registrada dentro da grade medida durante o $k^{ème}$ período de tempo;

ρ_{kij} – Média da carga estimada para o período de tempo (k+1) dos servidores *Web* do domínio de classe de QoS «i», dentro de um *cluster* «j», calculada durante o $k^{ème}$ período de tempo;

$$\rho_{kij} = \left(\frac{\sum_{i=1;j=1}^{N_d;N_c} \rho_{ksij}}{N_{sij}} \right)$$

χ_{kj} – Média da carga estimada para o período de tempo (k+1) do *cluster* «j», calculada durante o $k^{ème}$ período de tempo;

$$\chi_{kj} = \left(\frac{\sum_{j=1}^{N_d} \rho_{ksij}}{N_d} \right)$$

$\chi_{k\ min}$ – Valor da carga estimada do *cluster* menos carregado durante o período de tempo (k+1):

$$\chi_{k\ min} = Min_{j=1}^{Nc} (\chi_{kj})$$

$\mathbf{M_{kij}}$ – Modo de trabalho do domínio de classe QoS «i» pertencente ao um *cluster* «j» durante o período de tempo (k+1) calculada a partir da função $f_{ac\text{-}mode}$:

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

$\boldsymbol{\rho_{kj\ min}}$ – Valor da carga estimada do domínio de classe de QoS «i» menos carregado pertencente ao *cluster* «j» menos carregado, durante o período de tempo (k+1):

$$\rho_{kj\ min} = Min_{i=1}^{Nd}(\rho_{kij})$$

$\boldsymbol{\rho_{ksij\ min}}$ – Valor da carga estimada do servidor *Web* menos carregado «s» dentro da classe de QoS «i», pertencente ao *cluster* «j», durante o período de tempo (k+1):

$$\rho_{ksij\ min} = Min_{s=1}^{Nsij}(\rho_{ksij})$$

O mecanismo é executado em quatro momentos:

Monitoramento das Cargas: assim como citado no tópico 4.4.3, em Gerenciamento dos Recursos de Processamento, a cada período de tempo "k" o *Grid Resource Menager* - GRM solicita a todos os clusters informações de cargas (identificação do *cluster*, classes de domínio, média das classes, servidores *Web* e seus coeficientes de reatividade). Em seguida, o GRM estima a carga da grade inteira gerando as médias de cada *cluster* (r_{kj}). Os dados são armazenados em memória e utilizados pelo componente *UDDI Grid Scheduler* para que, quando uma requisição chegar num certo período "k", o mecanismo G-DSAC possa ser executado.

Seleção do Cluster Menos Carregado: Ao iniciar o processo de descoberta, os parâmetros são enviados para o mecanismo com base nos valores de cargas estimadas e é selecionado o *cluster* menos carregado ($\chi_{k\ min}$). A partir deste momento é realizada a tomada de decisão de qual servidor atenderá a requisição com base na classe de serviço da requisição e no modo de trabalho dos domínios de classes do *cluster* selecionado.

Determinando os Modos de Trabalho: Os "modos de trabalho" ou "estados" podem alternar entre: modo compartilhado, modo exclusivo e modo saturado. Quando a classe de domínio está trabalhando no modo compartilhado, significa que durante certo período de tempo "k" o servidor *Web* possui recursos suficientes para processar as requisições de qualquer classe de QoS dentro da plataforma sem comprometer os acordos de SLAs de sua classe nativa, ou seja a classe que ele atende prioritáriamente.

Quando o domínio de classe passa para o modo exclusivo significa que os níveis de carga atingiram determinados valores que podem comprometer os acordos de SLAs. Ou seja, aceitando as requisições de outras classes, as requisições da "classe nativa" poderão ser recusadas já que os recursos alocados prioritariamente para ela estão sendo utilizados por outras classes. Logo, durante certo período de tempo o domínio de classe aceitará exclusivamente as requisições da mesma classe.

No momento que o domínio de classe passa para o modo saturado, quer dizer que ele não aceitará mais novas requisições, mesmo de sua "classe nativa", garantindo assim os acordos de SLAs das requisições que estão em processamento. O que significa que todos os recursos estão sendo utilizados para processar as requisições em andamento e que irá recusar novas.

A determinação dos modos de trabalho (compartilhado, exclusivo ou saturado) é realizada através da função condicional ($f_{ac\text{-}mode}$) e o resultado é associado aos servidores *Web* localizados pelo processo de descoberta.

Resultado da Requisição: Para finalizar o processo de descoberta, o algoritmo abaixo informa ao cliente a URL do servidor *Web* que irá atender melhor a requisição.

Se ($\rho_{kj\,min} \leq R_i^{ac}$) e ($M_{km} = 0$)
  Enviar URL do servidor Web menos carregado ($\rho_{ksij\,min}$) da classe de domínio « m » ;
Senão
  Se ($\rho_{kij} \leq R_i^{ac}$)
    Enviar URL do servidor Web menos carregado ($\rho_{ksij\,min}$) da classe de domínio « i » ;
  Senão
    Enviar sinalização de erro;
  Fim se
Fim se

4.4.3. O Elemento C-DSAC (*Cluster - DiffServ Admission Control*)

O C-DSAC (*Cluster – DiffServ Admission Control*) permanece localizado nos clusters que fazem parte da Grade Computacional e é composto pelos seguintes elementos:

Web Server **(Servidor Web)**: Responsável por armazenar *Web services*, processar as requisições e hospedar o agente monitor de carga que calcula o coeficiente de reatividade [SERRA; BARROSO; BOUDY, 2004];

Cluster Resource Manager: Responsável por se comunicar com os agentes monitores, obter as informações de cargas e gerar as médias de carga para cada domínio de classe.

Diversos serviços podem ser armazenados nos servidores *Web*, descritos e disponibilizados aos clientes que acessam a grade computacional. Os serviços *Web* consistem de processos de negócios online realizados por aplicativos (ou componentes de software) compartilhados entre empresas, ou entre empresas e pessoas, sem interferência humana direta. Um exemplo bastante simples que pode ser citado é o serviço de CEP (Código de Endereçamento Postal) fornecido pelos Correios.

Cada servidor *Web* possui um agente monitor que monitora a sua carga. O agente monitor usa um coeficiente de reatividade [SERRA; BARROSO; BOUDY, 2004] que fornece uma idéia da tendência da carga do servidor. Esta técnica mede o tempo que uma tarefa espera para ter acesso ao tempo de processamento. O tempo de espera depende diretamente da carga do servidor.

O componente *Cluster Resource Manager* monitora as cargas do *cluster* como um todo coletando as tendências de carga de cada servidor *Web* e o domínio de classe a qual pertence. Logo em seguida, é obtida a média das cargas dos servidores por domínio de classe.

4.5. Arquitetura Proposta VS Trabalhos Relacionados

O problema da Qualidade de Serviço em ambientes de grades computacionais é um tema atualmente tratado em diversos projetos de pesquisa [MOHAMMAD; ANALOUI, 2006] [WAHIB et al, 2008] [HAO, YANG, LIN, ZHAI, 2006]. Dentre os trabalhos encontrados, alguns possuem pontos em comum com o trabalho aqui proposto. O mais próximo à proposta da arquitetura G-DSAC se encontra no projeto Grid-JQA.

Os autores L. Mohammad Khanli e M. Analoui da Universidade de Ciência e Tecnologia do Irã (Iran University of Science and Technology - IUST) desenvolveram uma estrutura que eles denominaram como Grid-JQA [MOHAMMAD; ANALOUI, 2006], a qual visa atender aos requisitos de QoS (tais como: mapeamento de requisitos de QoS para as capacidades de recursos, estabelecimento de SLA, parâmetros de monitoramento, entre outros) gerenciando os serviços de um banco de dados ativo, fornecendo reservas e gestão fim-a-fim em diferentes tipos de recursos (CPU, memória, disco, rede, etc.) e incentivando aos usuários da grade a especificar as necessidades de QoS. O trabalho mostra como objetivo dar à grade computacional os benefícios da QoS e da diferenciação de serviços através da correspondência entre os recursos solicitados e os recursos que melhor atendem ao pedido do usuário.

Entretanto, o Grid-JQA não possui a mesma visão de diferenciação e qualidade de serviço que o G-DSAC. A visão do Grid-JQA é de que os pedidos dos clientes possam ser atendidos de acordo com o tipo de recurso (serviços), ou seja, a “correspondência dos recursos”. Funciona da seguinte forma: é realizada uma correlação entre o recurso solicitado e o recurso armazenado no corretor e daí pode-se tomar duas decisões: ou é solicitada ao cliente uma negociação para decidir qual recurso utilizar, ou o próprio corretor de serviços faz a equivalência e devolve a que considerar melhor para atender o pedido. Outra particularidade desta arquitetura é que ela permite ao usuário definir seus parâmetros de QoS no momento do pedido.

Algumas características diferenciam o G-DSAC do Grid-JQA. No G-DSAC existe a possibilidade de se definir níveis diferentes de prioridade para utilização dos recursos de processamento. Isto não é tratado pelo Grid-JQA. Por outro lado, o Grid-JQA permite ao usuário definir seus parâmetros de QoS no momento do pedido enquanto que, no G-DSAC, os mesmos são definidos no momento da configuração da plataforma. No G-DSAC visando se adequar às arquiteturas em grades orientadas a serviços, o tratamento da QoS foi inserido no processo de descoberta do serviço *Web*.

Outro trabalho que aborda o tema é o modelo SOAG - *Service Oriented Architecutered Grid* [WAHIB et al, 2008]. O trabalho tem como objetivos: A modelagem da arquitetura Grid usando uma estrutura de SOA; Propor um mecanismo através do qual o desenvolvedor do aplicativo possa definir atributos de QoS para os serviços da camada de aplicação; Propor um método para a agregação dos atributos definidos de QoS na camada de aplicação com os atributos de QoS básicos a serem utilizados na programação dos trabalhos submetidos à aplicação Grid.

O que o SOAG foca é a implementação do modelo para Meta Heurísticas Grid – MHGrid, cuja finalidade é resolver problemas de otimização global com uma estrutura de caixa-preta. Os autores desenvolveram camadas middlewares (superiores e inferiores) que se incorporam ao modelo de grades computacionais e que são responsáveis pelo compartilhamento dos recursos e entregando-os de forma clara com um poder de processamento elevado para as aplicações que rodam em cima das mesmas. O G-DSAC não faz ajustes ao modelo de grades, mas desenvolve uma nova arquitetura levando em consideração aspectos das arquiteturas em grades orientadas a serviços.

Por fim, o trabalho desenvolvido pelos autores: Weidong Hao, Yang Yang, Chuang Lin, Zhengli Zhai (Pequim) cujo título é *QoS-aware Scheduling Algorithm Based on Complete Matching of User Jobs and Grid Services* [HAO, YANG, LIN, ZHAI, 2006], trata do modelo de escalonamento do Grid Services baseada na teoria dos grafos, que permitem uma melhor adaptação de tarefas paralelas para execução condicional cujo tempo de execução é limitado. Os autores estabeleceram um serviço de escalonamento e um serviço de sensor independente do provedor de recursos e do consumidor de recursos. O serviço de escalonamento busca a

correspondência adequada entre o consumidor e o prestador de serviços e o serviço de sensor monitora o estado de execução dos recursos e empregos em tempo real.

Mesmo realizando monitoramento de cargas e tratando a Qualidade de Serviço relacionada aos tempos de execução de serviços *Web*, alguns pontos distinguem o G-DSAC ao QoS-aware. O QoS-aware trabalha com grafos e matrizes e com a gestão dos recursos da largura de banda de rede e armazenamento de disco rígido. O G-DSAC utiliza como parâmetro para a medição da carga o "coeficiente de realividade" que leva em consideração a disponibilidade de processamento.

Após analisar os trabalhos citados acima e outros relacionados à QoS em ambientes distribuídos, foi observado como diferencial do trabalho aqui proposto a preocupação com a diferenciação da QoS ofertada aos clientes da grade. O G-DSAC propõe a concepção de uma solução voltada para grades computacionais que apóiem os procedimentos de garantia dos SLAs (*Service Level Agreements*) estabelecidos com os clientes de forma a otimizar o uso dos recursos de processamento disponibilizados, além de possibilitar a diferenciação de serviços no que diz respeito aos tempos de resposta oferecidos aos clientes da rede, usuários finais e aos serviços consumidores.

4.6. Conclusões

Neste capítulo foi apresentada a proposta de extensão da arquitetura WS-DSAC [SERRA, 2005] com objetivo de adequá-la aos novos paradigmas da computação distribuída e da engenharia de software, tratando aspectos relativos às aplicações desenvolvidas para Arquiteturas em Grade Orientadas a Serviços [SRINIVASAN; TREADWELL, 2005]. A nova arquitetura, denominada G-DSAC, introduz um bloco de funcionalidades que é responsável pela autenticação, classificação, publicação e descoberta de serviços levando em consideração aspectos ligados à QoS oferecida pela grade computacional. Com a modificação proposta na Arquitetura WS-DSAC é possível: realizar o balanceamento de cargas em três níveis (Grade, *Clusters* e Domínios de Classe), permitir a diferenciação de serviços em relação à utilização dos recursos de processamento disponíveis na grade e utilizar de forma eficaz esses recursos realocando-os de forma dinâmica entre classes diferentes de serviços.

V

Protótipo G-DSAC

Implementação e Avaliação

5.1. Introdução

Visando avaliar a capacidade do mecanismo G-DSAC em realizar os objetivos por ele propostos, foi implementado um protótipo funcional da plataforma com base na nova arquitetura concebida e apresentada no capítulo anterior.

Para a implementação do protótipo da plataforma G-DSAC foi utilizada a tecnologia Java. Um dos motivos de se utilizar o Java foi o fato da plataforma WS-DSAC ter sido desenvolvida nesta tecnologia. A plataforma WS-DSAC foi modificada a fim de lidar com os novos paradigmas ligados aos *Web Services.* Para isto, como apresentado no Capítulo IV, foram concebidos: um componente de monitoramento (*Grid Resource Manager)*, um corretor de serviços denominado (*UDDI Grid Scheduler*), um elemento de classificação (*Classifier Mechanism)* e um de autenticação (*Authentication Mechanism)* utilizando a linguagem de programação Java.

Neste capítulo são apresentados as tecnologias utilizadas na implementação da plataforma G-DSAC, os passos executados para o desenvolvimento do protótipo e as funcionalidades de cada elemento, o ambiente de teste e sua configuração, e finalmente os experimentos realizados e os resultados obtidos.

5.2. Tecnologias de Implementação

Para os processos de implementação e avaliação do protótipo da arquitetura G-DSAC foram adotadas algumas tecnologias. O Apache Tomcat [APACHE, 2008] foi utilizado tanto nos fornecedores (servidores *Web*) como no corretor de serviços da grade. No lado fornecedor foi também utilizado o Framework Apache Axis a fim de permitir a geração automática dos *Web Services* (.wsd), de seus arquivos de descrição (.wsdl) e prover um container para os serviços. No corretor de serviços da grade o Tomcat foi utilizado como *Container* integrado ao jUDDI, tornando-se um servidor UDDI com a capacidade de realizar os processos de descoberta e publicação dos *Web Services* disponibilizados pelos fornecedores. As tecnologias Java, Java RMI e o JEE Servlet foram utilizadas respectivamente como linguagem de programação para a implementação da plataforma, suporte de comunicação entre os objetos de monitoramento do G-DSAC e para criar interfaces para a administração dos recursos da grade. Já o Eclipse – Galileo foi utilizado como o ambiente de programação e com ele foram utilizados dois *plugins*: *JBoss Tools e JBoss Seam*.

Para implementar o corretor de serviços foram utilizados: jUDDI para criar um servidor UDDI e, o banco de dados MySQL Server para armazenar as descrições de *Web Services* publicados.

Uma descrição detalhada sobre as tecnologias foi disponibilizada no Anexo I deste documento.

5.3. A Plataforma (protótipo) G-DSAC

O desenvolvimento do protótipo da plataforma G-DSAC foi realizado em etapas. Cada etapa se constituiu na construção de diferentes componentes que ao final foram agregados de forma a viabilizar a realização de experimentos.

5.3.1. Escolha das Tecnologias

Como já citado no tópico 5.2, algumas tecnologias foram selecionadas de forma a viabilizar a implementação do protótipo da plataforma. A escolha de cada tecnologia foi

realizada conforme a necessidade para a implementação de cada componente. Na Figura 5.1 são apresenta as tecnologias selecionadas.

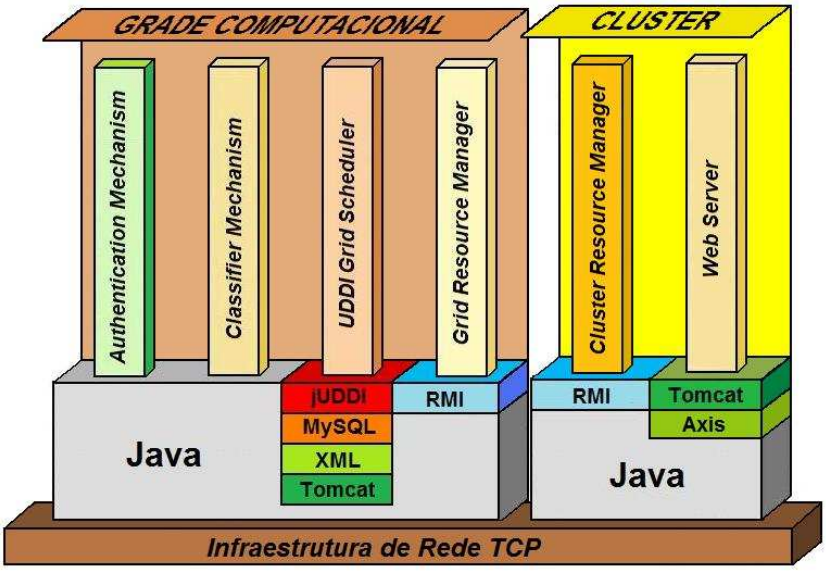

Figura 5.1: ***Tecnologias utilizadas para a implementação da protótipo G-DSAC***

Na Figura 5.1 são apresentados os componentes da plataforma (ver Capítulo IV) e quais tecnologias foram escolhidas de acordo com as características de cada um deles (ver Anexo I).

5.3.2. Modificações da Arquitetura WS-DSAC

A arquitetura WS-DSAC precisou ser modificada para que houvesse uma melhor integração com a extensão G-DSAC que tem por objetivo contemplar a tecnologia de *Web Services*. A implementação das modificações passou pelas seguintes fases (Figura 5.2):

1. Adaptar a plataforma para, além de servlets, hospedar *Web Services;*

2. Remover o componente *Cluster Gateway* e transferir suas funcionalidades de monitoramento, cálculos da média e da carga estimada dos servidores *Web* para o elemento *Global Resource Manager* do componente *Class Switch;*

3. Remover *os elementos Message Object Gateway e* o de Classificação;

4. Modificar o algoritmo WS-DSAC foi também modificado para atender a arquitetura G-DSAC transferido-o para o componente *UDDI Grid Sheduler* do bloco introduzido;

5. Implementar a arquitetura WS-DSAC modificada nos clusters e renomeá-la para C-DSAC (*Cluster - DiffServ Admission Control*).

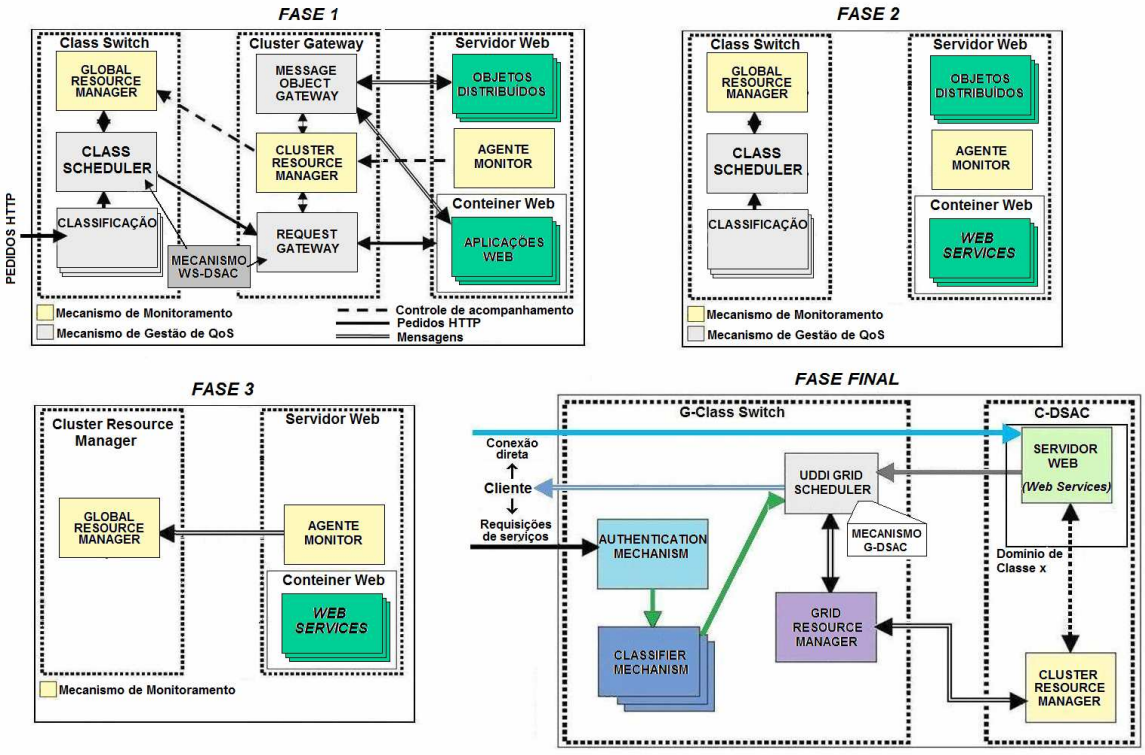

Figura 5.2: ***Transição da arquitetura WS-DSAC para arquitetura G-DSAC***

Na Figura 5.2 é apresentada a sequência das mudanças da arquitetura WS-DSAC até a sua integração com a extensão G-DSAC. A primeira fase mostra a plataforma original. Já nas segunda e terceira fases são apresentadas as alterações realizadas conforme citado nos itens dois, três e quatro. Na fase final é mostrado o resultado da integração com a extensão G-DSAC.

5.3.3. Implementação do Componente *Grid Resource Manager* (GdRM)

Como já explanado no Capítulo IV, o GRM possui duas funções distintas, mas que ao mesmo tempo estão interligadas. São elas: a realização do monitoramento das cargas dos clusters pertencentes à grade computacional, e a disponibilização para o mecanismo das informações relativas à localização e à carga dos servidores *Web* onde o serviço se encontra disponível. As

duas funções foram implementadas separadamente e depois incorporadas formando o componente *Grid Resource Manager*.

Foi utilizada a tecnologia Java RMI [SUN, 2008] (*Remote Method Invocation*) como suporte de comunicação entre os objetos de monitoramento de carga da plataforma.

5.3.4. Implementação do Componente *UDDI Grid Scheduler*

Como visto, também no capítulo 4, o *UDDI Grid Scheduler* é composto por um elemento (Corretor de Serviços), três métodos (Processo de Descoberta, Processo de Publicação e Processo de Envio) e um mecanismo (Mecanismo G-DSAC).

Foram utilizadas para esta etapa as seguintes tecnologias:

- **Corretor de Serviço:** MySQL Server, MySQL Query Browser, MySQL-Connector, Apache Tomcat e o Java UDDI – jUDDI.

- **Processos de Publicação, Descoberta e Envio:** Java, XML e Java *Web* Service.

- **Mecanismo G-DSAC:** Java.

A implementação do mecanismo tomou como base o algoritmo WS-DSAC (ver Capítulo IV). Foram incorporadas novas variáveis e novos métodos com o objetivo de adequá-lo a nova plataforma. Além de tratar as informações de carga e de classe, também são tratadas as informações sobre a carga dos clusters.

5.3.5. Implementação dos Componentes *Authentication Mechanism* e *Classifier Mechanism*

Os componentes de Autenticação (*Authentication Mechanism*) e Classificação (*Classifier Mechanism*) foram implementados utilizando-se a linguagem de programação Java.

Concluídas as modificações e a implementação dos componentes, a fase seguinte consistiu em criar um ambiente de testes visando realizar experimentos para avaliação da plataforma.

5.4. Ambiente de Teste

A fim de avaliar a capacidade da plataforma em cumprir com os objetivos propostos para ela foi montado um ambiente de teste real, descrito a seguir.

5.4.1. Arquitetura de Hardware e Software

A arquitetura de hardware e software consiste em um cenário heterogêneo e os equipamentos utilizados que foram selecionados para compor o ambiente de teste possuem as seguintes configurações (Figura 5.3):

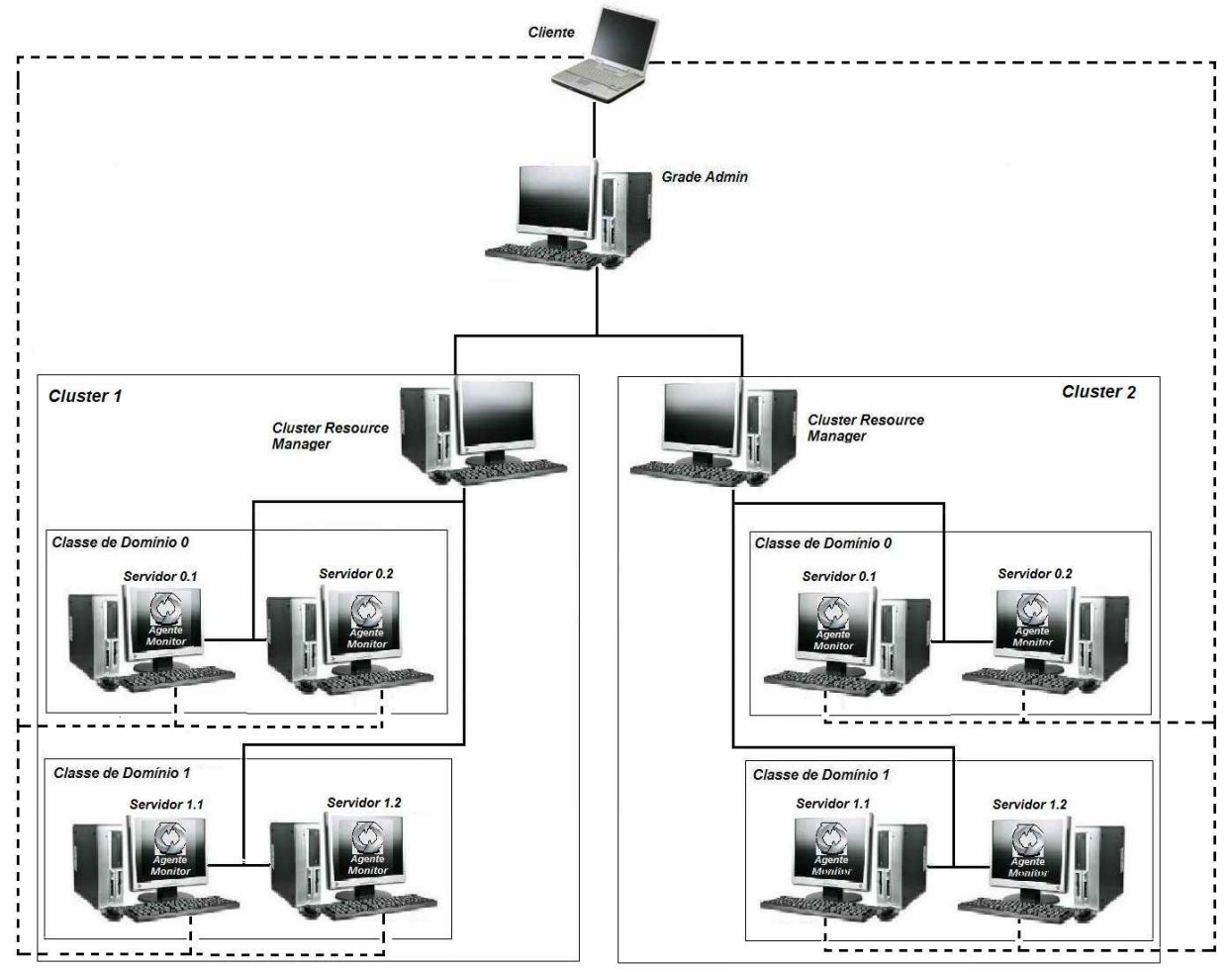

Figura 5.3: ***Ambiente de Teste***

- Servidores *Web*: PCs genéricos contendo processadores Core 2 Duo – 1.8 Ghz, 3Gb de RAM DDR, placa de rede Realtek RTL8169/8110 Family Gigabit Ethernet NIC e sistema operacional Windows XP.

- *Cluster Resource Manager*: PCs genéricos contendo processadores Core 2 Duo – 1.8 Ghz, 3Gb de RAM DDR, placa de rede VIA Rhine II Fast Ethernet Adapter e sistema operacional Windows XP.

- Grade Admin: PC genérico contendo processadores Core 2 Duo – 1.8 Ghz, 3Gb de RAM DDR, placa de rede VIA Rhine II Fast Ethernet Adapter e sistema operacional Windows XP.

- Cliente: Notebook contendo processadores Intel Core 2 Duo – 2 Ghz, 3Gb de RAM DDR 2, placa de rede Broadcom 802.11g e sistema operacional Windows Vista 32 bits.

Os Agentes Monitores foram instalados nos servidores *Web* e como "container" foram utilizadas as tecnologias Tomcat v. 6.0.18 e Axis v.1.4.

5.5. Experimentos e Resultados Obtidos

Os experimentos foram realizados visando avaliar a eficácia de dois principais objetivos do mecanismo ("Diferenciação de Serviços em momentos críticos" e "Equidade em momentos de baixa carga") e avaliar o desempenho em relação ao tempo de resposta para o consumidor final.

Para realizar os experimentos foram definidos dois cenários. No primeiro cenário (descrito no tópico 5.5.1) é mostrada uma situação de sobrecarga onde o mecanismo G-DSAC atua visando realizar a diferenciação de serviço. No segundo cenário (descrito no tópico 5.5.2) é mostrada uma situação de baixa carga em que o mecanismo G-DSAC é capaz de promover a equidade no uso dos recursos disponíveis.

Além dos cenários foi definido que dentro de cada *cluster* pertencente à grade, seriam especificados dois domínios de classes: classe "0" (mais prioritária) e classe "1" (menos prioritária). Os parâmetros para cada classe foram determinados como mostrado na Tabela 5.1.

Web Services foram replicados nos servidores *Web* pertencentes tanto ao domínio classe “0” quanto ao domínio de classe “1”.

	Classe 0	**Classe 1**
Rac	300ms	600ms
Rmax	210ms	420ms

Tabela 5.1: ***Parâmetros específicos para as classes de QoS***

5.5.1. Experimento 1 - A Garantia da Diferenciação de Serviços em Momentos Críticos

Este experimento visa avaliar a capacidade de controle de admissão e diferenciação de serviços da plataforma em momentos em que existem muitas requisições enviadas aos *Web Services* disponibilizados, ou seja, em momentos críticos de cargas.

Para a composição do cenário em que os servidores *Web* estão trabalhando em um modo crítico de carga, ou seja, em um modo exclusivo ou saturado, foi utilizada uma aplicação para geração e envio de requisições, denominada *Web Stress* com o objetivo de sobrecarregar os servidores com um nível de carga constante. Em seguida o programa que simula aplicações consumidoras foi executado gerando requisições em intervalos de tempo constantes, pertencentes a classes específicas, conforme apresentado na Tabela 5.2:

Fases	**No. Requisições**	**Intervalo de Tempo**	**Classes**
1a.	20	1 s	0
	20	1 s	1
2a.	40	2 s	0
	40	2 s	1

Tabela 5.2: ***Valores dos Parâmetros para o Experimento 1***

Durante os experimentos foram gravadas informações relevantes para as análises e para a verificação da eficácia da arquitetura proposta no que diz respeito ao controle de admissão e a diferenciação de serviços.

1ª. Fase

Para a primeira fase foi realizado o experimento simulando o envio de 20 (vinte) requisições com o nível de classificação “0” em um intervalo de tempo de 1 (um) segundo

entre elas e em seguida o envio de 20 (vinte) requisições com o nível de classificação “1” em um intervalo de tempo de 1 (um) segundo. Após o experimento foi realizada uma análise para verificar o comportamento das cargas e o desempenho em relação ao tempo de resposta.

Em relação ao comportamento das cargas, o que se pode observar é que, dentro dos experimentos, o mecanismo foi capaz de equilibrar as cargas das classes dos serviços disponíveis tomando como prioridade a classe nativa, não ultrapassando os parâmetros pré-estabelecidos para as respectivas classes, ou seja, ele foi capaz de manter a variação da carga dentro dos limites especificados para cada classe (Tabela 5.1). Note que nas Figuras 5.4 e 5.5 é mostrado o comportamento das cargas das classes de domínio de QoS “0” e “1”, mensuradas respectivamente no *cluster* 1 e no *cluster* 2, durante o atendimento simultâneo das requisições de classe.

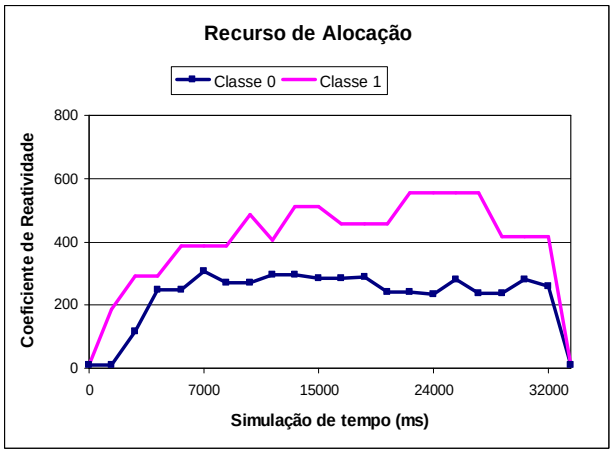

Figura 5.4: ***Comportamento da carga dos domínios de classes QoS “0” e “1” – Cluster 1***

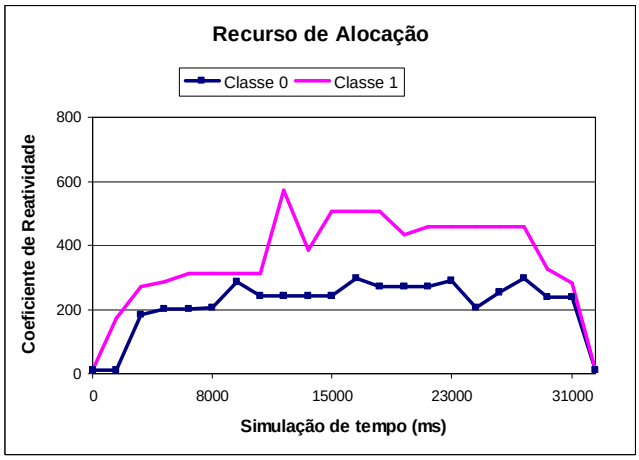

Figura 5.5: ***Comportamento da carga dos domínios de classes QoS “0” e “1” – Cluster 2***

O que foi observado no que diz respeito ao desempenho em relação ao tempo de resposta é que a plataforma conseguiu fornecer aos clientes a resposta da requisição do serviço de forma eficiente mesmo atuando em uma situação de sobrecarga levando em consideração o nível de classe de QoS. Na Figura 5.6 é apresentada a medição do tempo de resposta que levou cada uma das requisições atendidas pelos servidores.

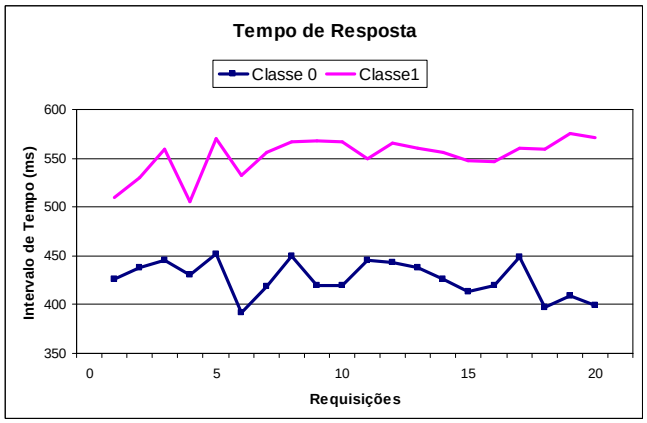

Figura 5.6: ***Medição do Tempo de Resposta para as requisições de serviços***

2ª. Fase

Para a segunda fase foi realizado o experimento simulando o envio de 40 (quarenta) requisições com o nível de classificação "0" em um intervalo de tempo de 2 (dois) segundos entre elas e em seguida o envio de 40 (quarenta) com o nível de classificação "1" em um intervalo de tempo de 2 (dois) segundos. Após o experimento foi realizada uma análise para verificar o comportamento das cargas e o desempenho em relação ao tempo de resposta.

Assim como na primeira fase, o que se pode observar é que, dentro dos experimentos, em relação ao comportamento da carga, o mecanismo também foi capaz de equilibrar as cargas das classes dos serviços disponíveis tomando como prioridade a classe nativa dentro dos limites estabelecidos para cada classe [Tabela 5.1]. Note que nas Figuras 5.7 e 5.8 é mostrado o comportamento das cargas das classes de domínio de QoS "0" e "1", mensuradas respectivamente no *cluster* 1 e no *cluster* 2, durante o atendimento simultâneo das requisições de classe.

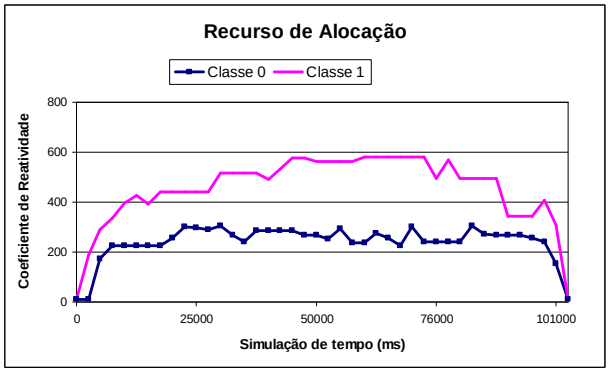

Figura 5.7: ***Comportamento da carga dos domínios de classes QoS "0" e "1" – Cluster 1***

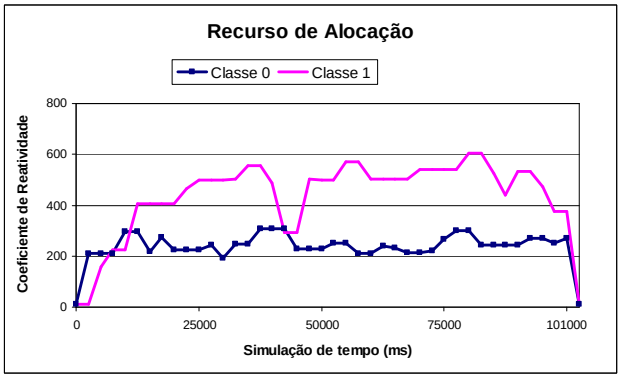

Figura 5.8: ***Comportamento da carga dos domínios de classes QoS "0" e "1" – Cluster 2***

Em relação ao tempo de resposta para esta segunda fase, houve uma pequena aproximação entre os tempos da Classe “0” e da Classe “1”, contudo o intervalo de tempo não ultrapassou o limite de 600 ms, o que se conclui que a plataforma permaneceu fornecendo aos clientes um tempo eficiente mesmo atuando em uma situação de sobrecarga levando em consideração o nível de classe de QoS.

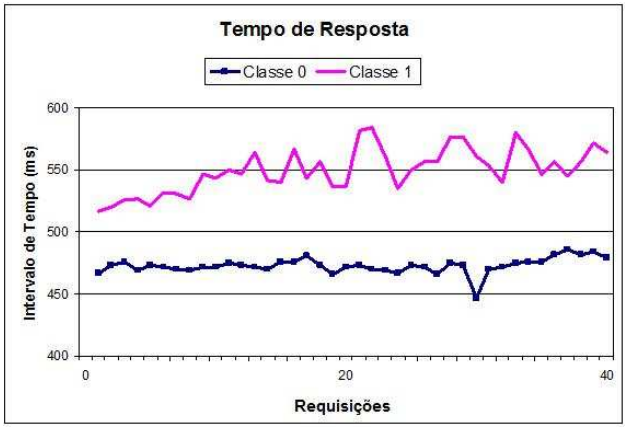

Figura 5.9: ***Medição do Tempo de Resposta para as requisições de serviços***

5.5.2. Experimento 2 - A Garantia da Equidade em Momentos de Baixa Carga

Este experimento ilustra a capacidade do mecanismo em prover equidade (“*fairnes*”), ou seja, promover uma igualdade no emprego dos recursos em momentos onde não exista uma grande quantidade de requisições a serem atendidas.

Para a composição do cenário em que os servidores *Web* estão trabalhando com baixa carga, ou seja, em um modo compartilhado, bastou deixá-los livres de requisições e não utilizar nenhum recurso para sobrecarregá-los o que tornou a utilização do *Web Stress* dispensável. Em seguida, o programa que simula aplicações consumidoras foi executado gerando requisições em intervalos de tempo constantes pertencentes a classes específicas, conforme apresentada na Tabela 5.3.

No. Requisições	Intervalo de Tempo	Classes
40	2 s	0
40	2 s	1

Tabela 5.3: ***Valores dos Parâmetros para o Experimento 2***

Da mesma forma que no experimento 1, foram gravadas informações relevantes durante o segundo experimento para uma análise e para a verificação da eficácia da arquitetura proposta no que diz respeito a equidade

O que se pode observar em relação ao comportamento da carga é que o mecanismo possui a capacidade de compartilhar os recursos disponíveis com todas as requisições de maneira justa e independente da classe de QoS em momentos de baixa carga. Este resultado foi obtido pelo fato de que os servidores *Web*, pertencentes aos *clusters* que integram a plataforma da grade, estavam trabalhando em modo compartilhado. Na Figura 5.10 é apresentado o caso de 40 (quarenta) requisições com o nível de classe "0" enviadas em intervalo de 2 (dois) segundos. Observe que os servidores de domínio de classe de QoS "0" e de classe de QoS "1" tratam de forma compartilhada os recursos disponíveis com todas as requisições com o nível de classe "0". O segundo caso (Figura 5.11) de 40 (quarenta) requisições com o nível de classe "1" enviadas em intervalo de 2 (dois) segundos, apresenta resultado semelhante, ou seja, os servidores de domínio de classe de QoS "0" e de classe de QoS "1" tratam de forma compartilhada os recursos disponíveis com todas as requisições com o nível de classe "1". O que se conclui é que: as requisições com níveis de classe diferentes foram tratadas de forma equitativa em relação ao uso dos recursos de processamento.

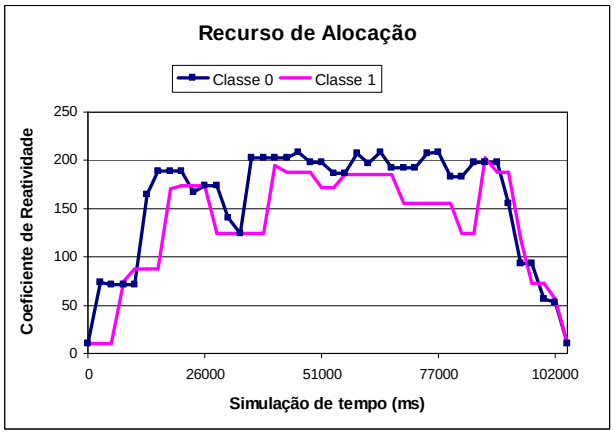

Figura 5.10: ***Comportamento da carga dos domínios de classes QoS "0" das requisições de classe "0"***

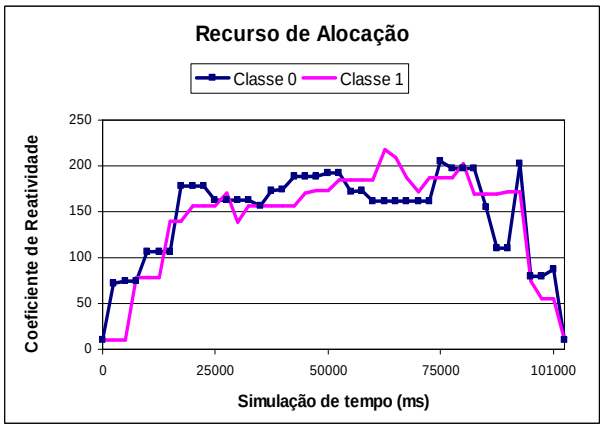

Figura 5.11: ***Comportamento da carga dos domínios de classes QoS "1" das requisições de classe "1"***

Neste segundo experimento, o desempenho em relação ao tempo de resposta para as requisições tanto pertencentes à classe de QoS "0" como à classe de QoS "1", apresentou um resultado bastante satisfatório. O intervalo de tempo não ultrapassou 460 ms o que se distingue do primeiro experimento que apresentou limite máximo de tempo aproximadamente 600 ms. Note que na Figura 5.12 é apresentada uma proximidade entre os tempos de resposta entre as classes "0" e "1".

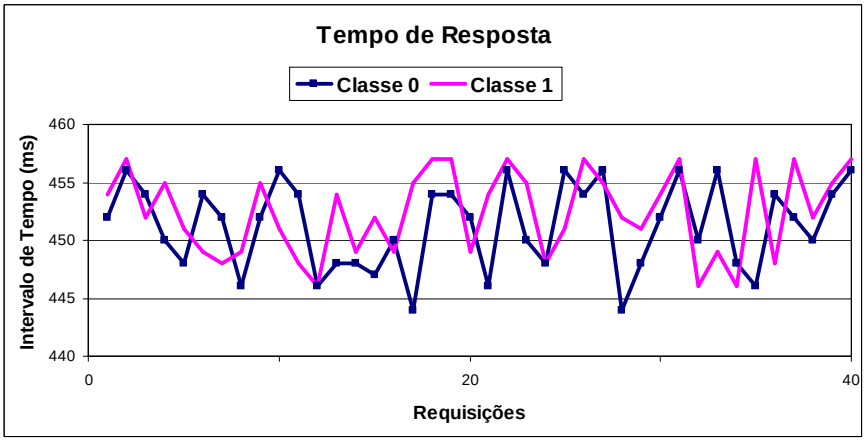

Figura 5.12: ***Medição do Tempo de Resposta para as requisições de serviços***

5.6. Conclusões

Neste capítulo foram apresentadas as tecnologias utilizadas na implementação da plataforma G-DSAC, a descrição do protótipo com as funcionalidades de cada elemento, o ambiente de teste, alguns dos experimentos realizados e os resultados obtidos.

Foram apresentados os experimentos realizados (Diferenciação de Serviços e Equidade) em que os resultados obtidos mostraram que, assim como o mecanismo WS-DSAC, o mecanismo G-DSAC é capaz de realizar o controle de admissão, assegurando os contratos de QoS instituídos para cada domínio de classe, dentro de um cenário crítico onde há muitas requisições, bem como a garantia da equidade ("*fairnes*") no uso dos recursos em momentos de baixa utilização. De forma que se pode concluir que a arquitetura atende ao propósito de prover uma solução voltada para grades computacionais que é capaz de garantir os SLAs estabelecidos com os consumidores de serviços utilizando de forma otimizada os recursos de processamento disponibilizados na grade. Proporcionando ainda, a possibilidade de diferenciação de serviços no que diz respeito aos tempos de resposta oferecidos aos clientes, usuários finais e aos serviços consumidores.

VI
Conclusão

6.1. Considerações Finais

Nesta dissertação foi apresentada uma proposta de extensão da arquitetura WS-DSAC (ver Capítulo III). Esta extensão, denominada G-DSAC, foi concebida com foco nas arquiteturas em grades computacionais orientadas a serviços. Ela compreende a concepção de uma solução voltada para grades computacionais [NASSIF; NOGUEIRA, 2007] que é capaz de garantir os SLAs (*Service Level Agreements*) estabelecidos com os consumidores de serviços utilizando de forma otimizada os recursos de processamento disponibilizados na grade. A solução permite ainda a diferenciação de serviços no que diz respeito aos tempos de resposta oferecidos aos usuários finais e aos serviços consumidores.

Na concepção desta arquitetura, foram levados em consideração aspectos importantes relacionados às arquiteturas em grades orientadas a serviços tais como: os pontos de estrangulamento, o balanceamento de cargas e a necessidade de diferenciação da QoS em relação à utilização dos recursos dentro da Grade.

A Arquitetura G-DSAC consiste em dois blocos (ver Capítulo IV): bloco da Extensão e bloco do *Cluster*. O bloco da extensão possui as funções de monitoramento da grade, descoberta e publicação de *Web Services* e autenticação e classificação de requisições. O bloco do *Cluster* possui as funcionalidades de gestão dos recursos (monitoramento e estimativa de cargas dos servidores). A Arquitetura G-DSAC contempla aspectos de uma arquitetura em grade orientada a serviços tomando como base a arquitetura WS-DSAC. Assim ela permite: a descoberta e publicação de serviços *Web*, o controle de admissão de requisições, o balanceamento de cargas e a oferta de diferentes níveis de QoS.

Para a implementação do protótipo da plataforma G-DSAC foi utilizada a tecnologia Java (ver Capítulo V). A plataforma WS-DSAC foi adaptada de forma a disponibilizar *Web Services*. Para isto, foram implementados: o componente de monitoramento (*Grid Resource Manager*), o corretor de serviços (*UDDI Grid Scheduler*), o de classificação (*Classifier Mechanism*) e o de autenticação (*Authentication Mechanism*) utilizando-se tecnologias Java. A fim de avaliar a capacidade da plataforma em atender os objetivos propostos, foi montado um ambiente real de testes composto por uma plataforma de hardware e software heterogênea.

Um conjunto de experimentos foi realizado para avaliar a capacidade da plataforma em atingir os objetivos por ela propostos e alguns destes experimentos foram apresentados para ilustrar a capacidade da plataforma em prover o balanceamento de cargas, garantir uma diferenciação de serviços e utilizar de forma equitativa os recursos de processamento disponibilizados na grade. Para uma melhor análise dos dados, os experimentos apresentados foram divididos em dois grupos: “A Garantia da Diferenciação de Serviços em Momentos Críticos” e “A Garantia da Equidade em Momentos de Baixa Carga”.

As análises mostraram que a plataforma G-DSAC é capaz de realizar o controle de admissão, assegurando os contratos de QoS instituídos para cada domínio de classe pré-estabelecido dentro de um cenário crítico onde há muitas requisições. Ela também foi capaz de garantir a equidade (“*fairnes*”) no uso dos recursos em momentos de baixa utilização, ou seja, de baixa carga.

6.2. Contribuições

Um dos pontos críticos das arquiteturas em grade orientadas a serviços é a questão da gestão da QoS (*Quality of Service*) e o compartilhamento eficaz dos recursos de processamento disponibilizados.

Inicialmente, a contribuição deste trabalho consiste em realizar uma reflexão sobre os novos desafios impostos ao problema da gestão da QoS no nível de aplicação em ambientes distribuídos levando em consideração os avanços tecnológicos relativos à comunicação e aos novos paradigmas da engenharia de software.

Como fruto desta reflexão, o trabalho apresenta uma solução pouco intrusiva que trata do balanceamento de cargas, da gestão da QoS (medida pelo tempo de resposta das requisições) e da diferenciação de serviços em um ambiente de grade computacional orientada a serviços utilizando de forma otimizada os recursos disponíveis.

Os resultados apresentados servem de motivação para que os elementos de gestão da QoS explorados dentro da solução concebida possam ser também aplicados em outras arquiteturas em grade orientadas a serviços. Com isso as mesmas podem tratar de forma mais eficaz o uso dos seus recursos de processamento e podem também promover o balanceamento de cargas e a diferenciação de serviços.

6.3. Limitações do trabalho

Na solução arquitetural proposta, a inteligência do mecanismo G-DSAC foi introduzida no “Corretor de Serviços”, que é o responsável pela publicação e pela descoberta de todos os serviços disponibilizados na grade. Em termos arquiteturais isso pode se tornar um gargalo à medida que o número de *clusters* que compõem a grade aumenta. Para melhor avaliar o impacto de desempenho no processo de descoberta dos serviços, levando em consideração o G-DSAC, seria necessária a implementação de um ambiente de simulação que permitisse a análise de cenários compostos por uma grande quantidade de *clusters* e servidores dentro da grade. O que não foi possível de realizar dentro do escopo do presente trabalho.

O processo de monitoramento e estimativa de cargas da plataforma está distribuído no nível de *cluster* e no nível de grade. Mais uma vez, por questões relativas à necessidade de alto desempenho no processo de descoberta de serviços, as informações de carga precisam ser consolidadas no “Corretor de Serviços” permitindo a tomada rápida de decisão (algoritmo G-DSAC) para escolha do servidor da grade que atenderá a chamada. Faz-se também necessária uma análise, por meio de simulações, do impacto deste processo de monitoramento no tempo de descoberta dos serviços levando em consideração um grande número de servidores dentro da grade.

6.4. Trabalhos Futuros

Para permitir uma avaliação mais aprofundada da Arquitetura G-DSAC, principalmente em relação aos aspectos ligados ao impacto da introdução da mesma em relação ao desempenho do ambiente em Grade Orientada a Serviços, seria interessante a simulação de um ambiente composto por um número maior de servidores e *clusters* do que o que foi utilizado para a realização dos experimentos apresentados. Para isto, seria interessante a modelagem do ambiente utilizando-se, por exemplo, Redes de Petri. Isto permitiria a avaliação de um conjunto de propriedades da solução proposta bem como uma análise do impacto da mesma no desempenho de uma grade com um grande número de servidores.

Visto que a computação em nuvens segue princípios semelhantes aos da computação em grade, pois compartilha recursos de forma distribuída, seria interessante uma reflexão sobre a possibilidade de utilização da arquitetura G-DSAC em um ambiente de Nuvens Computacionais levando em consideração os aspectos ligados à QoS e à diferenciação de serviços.

Finalmente, outro trabalho interessante a ser realizado seria a análise do uso da Arquitetura G-DSAC em outros ambientes em grade como, por exemplo: o *Globus* [GLOBUS, 2010], o *OurGrid* [OURGRID, 2007] ou o *EuroGrid* [EUROGRID, 2004]. Esta análise permitirá avaliar o grau de independência da solução em relação a uma plataforma específica.

6.5. Publicação

Cedro, Daniela M., Souza, José N., Serra, Antonio B. "*Gestão da QoS em Arquiteturas de GRELHAS Computacionais Orientadas a Serviços*". 9ª Conferência da Associação Portuguesa de Sistemas de Informação - CAPSI 2009, Viseu – Portugal. Sociedade Científica: Associação Portuguesa de Sistema de Informação - APSI.

VII

Referências Bibliográficas

BUYYA; R. High Performance Cluster Computing. *Architectures & Systems.* Prentice Hall PTR. Australia, v. 1. jul. 1999.

MORRISON; R. S. Cluster Computing - Architectures, Operating Systems, Parallel Processing & Programming Languages. *General Public Licence*. abr. 2003.

FOSTER, I.; KELSSELMAN, C. The Grid2: Blueprint for a New Computing Infrastructure. *Morgan Kaufmann Publishers*. nov. 2004.

NASSIF, LILIAN; NOGUEIRA, JOSÉ. Grades computacionais: uma tecnologia para compartilhamento de recursos em rede. *Revista Fonte,* Belo Horizonte. v. 6. jun. 2007.

CEDRO, DANIELA M. *A Qualidade de Serviços em Nível de Aplicação em Clusters de Servidores Web*. 2004. Monografia (Graduação em Ciência da Computação) – Departamento de Computação, Faculdade Lourenço Filho, Ceará. 2004.

KEEN, MARTIN et al. *Patterns: Implementing an SOA Using an Enterprise Service Bus*. USA. IBM. Julho/2004.

TANENBAUM, ANDREW S.; STEEN, MAARTEN VAN. *Sistemas Distribuídos – Princípios e Paradigmas*. 2a. Edição. São Paulo. Prentice Hall PTR. 2007.

PUTTE et al. *Using Web Services for business integration*. USA. IBM. 2004.

FOSTER et al. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Disponível em <http://www.globus.org/alliance/publications/papers/ogsa.pdf>.

CASTRO M., DWYER M., RUMSEWICZ M. Load balancing and control for distributed World Wide Web servers, *IEEE International Conference on Control Applications*, v. 2. 1999.

CHERKASOVA L., PHAAL P., Session-based admission control: a mechanism for peak load management of commercial Web sites, *IEEE Transactions on Computers*, v. 51, p. 669 – 685. 2002.

ABDELZAHER T.F., SHIN K.G., BHATTI N., Performance guarantees for Web server end-systems: a control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems.* v. 13, p. 80 – 96. 2002.

CARPENTER B.E., NICHOLS K., Differentiated services in the Internet. *Proceedings of the IEEE.* v. 90, p. 1479 – 1494. 2002.

ROSEN E. C. et al., *Multiprotocol Label Switching Architecture*. Disponível em < http://www.javvin.com/protocol/rfc3031.pdf >.

SERRA, ANTONIO B., *La gestion de la QoS pour les services Internet au niveau de grappes de serveurs Web*. 2005. Tese (Doutorado em Computação) - Institut National dés Télécommunication (INT). France.

OLEJNIK R., BOUCHI A. AND TOURSEL B., "*An Object observation for a Java Adaptative Distributed Application platform*". IEEE PARELEC'02, 22-25. pp: 171-176. Setembro/2002.

SERRA, A., GAÏTI D., BARROSO G., BOUDY J., A load-balancing Distributed Platform based on Differentiated Services for a Telecare Application. *IEEE International Conference on Control Aplications*. 2004.

W3C., *Web Services Description Requirements*. Disponível em <www.w3.org>. 2002.

SUN., *Java Remote Method Invocation*. Disponível em <http:// java.sun.com / products / jdk / rmi/>. 2008.

APACHE., *The Apache Jakarta Project*. Disponível em <http:// jakarta.apache.org / tomcat / index.html>. 2008.

HURWITZ, JUDITH et al., *Service Oriented Architecture for Dummies*. 1o. Edição. Canadá. Wiley Publishing. 2001.

SRINIVASAN, LATHA; TREADWELL, JEM., *An Overview of Service Oriented Architecture, Web Services and Grid Computing*. HP software Global Business. 2005.

MARTINS, G et al., Provendo um Mecanismo de Alta Disponibilidade em Clusters de Servidores Web. *Journal of Computer Science*. USA. 2008.

MARQUES, CARLA et al., Uma Solução para Aumentar a Disponibilidade de Serviços em Clusters de Servidores Web Baseada em Sistemas Multiagentes. *V Simpósio Brasileiro de Sistemas de Informação*. Brasília. 2009.

BARRETO, M; ÁVILA, R; NAVAUX, P., The Multi-Cluster Model to the Integrated use of Multiple Workstation Clusters. *3rd Workshop on Personal Computer based Networks of Workstations*. Cancun. 2000.

SERRANO, ALBANO JOSÉ, *Infra-estrutura para a Computação Multi-Cluster em Ambiente Grid*. 2006. Dissertação (Mestrado em Computação) – Faculdade de Engenharia de Universidade do Porto, Portugal. 2006.

BARRETO, M; NAVAUX, P., Um Modelo para a concepção de Grades Computacionais baseadas em Clusters. *Workshop do Grupo de Processamento Paralelo e Distribuído*. Porto Alegre. 2003.

PINHEIRO, JOSÉ MAURÍCIO, *Afinal, o que é Qualidade de Serviços*. Disponível em <www.cliconnect.com.br/artigos/afinaloqueQualidadeServicos.html>. 2004.

LUCENA K. F. M. et al., Grade Computacional como alternativa para o processamento de Alto Desempenho em Pesquisa Tecnológica. *I Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica*. Natal. 2009.

TAGLIARI, FABIANE; CARVALHO, ELIAS CÉSAR, Grids Computacionais. *Revista Científica Campo Digital*, v.1, n.1, p. 3-10. Campo Mourão - PR. Março/2006.

BANDINI, MATHEUS et al., A Grid-QoS Decision Support System using Service Level Agreements. *XXIX Congresso da Sociedade Brasileira de Computação*. Bento Gonçalves. 2009.

CIRNE, WALFREDO; NETO, ELIZEU SANTOS, Grids Computacionais da Computação de Alto Desempenho a Serviços sob Demanda. *23º Simpósio Brasileiro de Redes de Computadores*. Fortaleza. 2005.

TEXEIRA, FABIANO COSTA, *Um Escalonador para Grades Computacionais utilizando Modelos Econômicos*. 2006. Dissertação (Mestrado em Computação) Instituto de Computação – Universidade Estadual de Campinas. Campinas. 2006.

SCORSATTO, GLAUBER, GrAMoS: *Serviço para Monitoramento de Acordos em Grid*. 2007. Dissertação (Mestrado de Computação). Departamento de Ciências da Computação – Universidade de Brasília. Brasília. 2007.

SENNA, CARLOS ROBERTO, "*GPO – Um Middleware para Orquestração de Serviços em Grades Computacionais*". Dissertação (Mestrado de Computação) – Universidade Estadual de Campinas. 2007.

GIRARDI, REUBEM ALEXANDRE. *Framework para coordenação e mediação de Web Services modelados como Learning Objects para ambientes de aprendizado na Web*. Dissertação de Mestrado. PUC-Rio. Agosto/2004.

MARQUEZAN, CLARISSA; CARISSIMI, ALEXANDRE; NAVAUX, PHILIPPE, "*Web Services para Computação de Alto Desempenho*". WSCAD2006. Ouro Preto – MG. Setembro/2006.

SERRA ANTONIO et al., "*Controle de Admissão e Diferenciação de Serviços em Clusters de Servidores Web*". BDBComp. Uberlândia – MG. Abril/2005.

MOHAMMAD KHANLI L., ANALOUI M., "*Grid-JQA a New Architecture for QoS-guaranteed Grid Computing System*". 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. França. Fevereiro/2006.

WAHIB M. et al., "*SOAG: Service Oriented Architectured Grids and Adoption of Application specific QoS Attributes*". 9th Grid Computing Conference – Japão. Setembro-Outubro/2008.

HAO W., YANG Y., LIN C., ZHAI Z. "*QoS-aware Scheduling Algorithm Based on Complete Matching of User Jobs and Grid Services*". China. Dezembro/2006.

OURGRID. "*OurGrid WebSite*". Disponível em <http://www.ourgrid.org>. 2007.

EUROGRID. "*EuroGrid - Application Testbed for European GRID computing*". Disponível em <http://www.eurogrid.org>. 2004.

GLOBUS. "*The Globus Alliance*". Disponível em <http://www.globus.org>. 2010.

Anexo I

Tecnologias de Implementação do G-DSAC

Para os processos de implementação e avaliação do protótipo da Arquitetura G-DSAC foram utilizadas as seguintes tecnologias:

Apache Tomcat 6.0.18

Descrição: Servidor *Web* Java, mais especificamente, um container de *servlets*. O servidor inclui ferramentas para configuração e gerenciamento, o que também pode ser feito editando-se manualmente arquivos de configuração formatados em XML.

Função na Arquitetura: O Tomcat foi utilizado tanto no lado fornecedor (Servidor *Web*), como no lado servidor. No lado fornecedor foi utilizado juntamente com o Framework Apache Axis a fim de criar um *Web Service* automaticamente e seu arquivo de descrição (.wsdl) além de criar um container para o lado fornecedor. Do lado servidor foi utilizado como *Container* integrado ao jUDDI, tornando-se um servidor UDDI com a capacidade de realizar os processos de descoberta e publicações dos *Web Services* disponibilizados pelos fornecedores.

Apache Axis 1.4

Descrição: Um *framework* de código aberto, baseado na linguagem Java e no padrão XML, utilizado para construção de *Web Services* no padrão SOAP. O Axis disponibiliza dois modos para "expor" os métodos de uma classe através de *web services*. O modo mais simples utiliza os arquivos JWS (Java *Web* Service) do Axis. O outro método utiliza um arquivo WSDD (*Web* Service Deployment Descriptor), que descreve com detalhes como serão criados os *web services* a partir dos recursos (classes Java) existentes.

Também é possível, através do Axis, gerar automaticamente o arquivo WSDL (*Web* Service Description Language). O WSDL contém a definição da interface dos *web services*

Função na Arquitetura: O Axis foi utilizado a fim de criar *Web Services* e gerar automaticamente arquivos WSDL que foram utilizados durantes os experimentos e as análises da arquitetura.

JBoss Tools

Descrição: *JBoss Tools* é um conjunto de plugins para o Eclipse onde pode trabalhar com o servidor de aplicação e frameworks da **JBoss, como Hibernate, Seam, jBPM, Drools, Richfaces, JSF,Struts, JPA etc** .

Função na Arquitetura: *JBoss Tools* foi instalado no Eclipse para o desenvolvimento da plataforma.

JBoss Seam

Descrição: JBoss Seam é um framework facilitador da programação para todos os componentes em uma aplicação *Web*. Focando a produtividade do programador e a escalabilidade da aplicação.

Função na Arquitetura: *JBoss Seam* foi instalado no Eclipse para o desenvolvimento da plataforma.

jUDDI 0.9rc

Descrição: O jUDDI é uma implementação Java open source da especificação UDDI (Universal Description, Discovery, and Integration) para *Web Services*. Sendo o jUDDI uma aplicação *Web* Java pode ser utilizado com qualquer servidor de aplicação ou mecanismo servlet que suporte a versão 2.1 ou superior do servlet API.

Função na Arquitetura: O jUDDI foi utilizado integrando-se ao JBoss de forma a criar um Servidor UDDI.

MySQL Server 6.0

Descrição: O **MySQL Server** é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês *Structured Query Language*) como interface. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo.

Função na Arquitetura: Armazenar as descrições das *Web Services* publicadas.

MySQL Query Browser 1.2.17

Descrição: O MySQL Query Browser é uma ferramenta gráfica fornecida pela MySQL AB para criar, executar e otimizar solicitações SQL além de administrar um servidor MySQL e analisar dados armazenados dentro de um Banco de Dados MySQL.

Função na Arquitetura: Administrar os dados do servidor MySQL.

Java 6

Descrição: Java é uma linguagem de programação orientada a objetos que não se prende a nenhuma arquitetura e a nenhuma empresa, é rápida e estável. Pode construir de simples aplicações a sistemas críticos que precisam de grande velocidade.

Função na Arquitetura: Linguagem utilizada para a implementação da plataforma.

RMI Java 6

Descrição: RMI ou Remote Method Invocation existe na tecnologia Java a partir da versão JDK 1.1, utilizada para criar programas aplicações distribuídas na tecnologia Java. Possibilitando ao programador invocar métodos remotos de outras Java Virtual Machines (Maquinas virtuais Java), que podem ou não existir em diferentes host.

Função na Arquitetura: Utilizado como suporte de comunicação entre os objetos da Grade Computacional.

JEE Servlet 5

Descrição: Uma classe Java que processa que processa dinamicamente processos e

respostas. O Servlet disponibiliza ao programador Java um interface para o servidor de aplicação ou *Web*, através de uma API.

Função na Arquitetura: Utilizado para criar interfaces entre os *cluster* e a grade e entre os clientes e a grade.

Eclipse - Galileo

Descrição: é uma IDE desenvolvida em Java, com código aberto para a construção de programas de computador. Hoje o Eclipse é a IDE Java mais utilizada no mundo. Possui como características marcantes o uso da SWT e não do Swing como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em plug-ins e o amplo suporte ao desenvolvedor com centenas de plug-ins que procuram atender as diferentes necessidades de diferentes programadores.

Função na Arquitetura: Utilizado para desenvolver as aplicações para o funcionamento da plataforma G-DSAC.