Douglas de Oliveira Balen

Sistema para Gerência Autonômica de Grades Computacionais

Florianópolis – SC

Fevereiro / 2009

Douglas de Oliveira Balen

Sistema para Gerência Autonômica de Grades Computacionais

Dissertação apresentada ao Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciências da Computação

Orientador:
Carlos Becker Westphall

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Florianópolis – SC

Fevereiro / 2009

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciências da Computação - Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciências da Computação.

Prof. Dr. Frank Siqueira
Departamento de Informática e Estatística- UFSC
Coordenador do PPGCC - UFSC

Prof. Dr. Carlos Becker Westphall
Departamento de Informática e Estatística - UFSC
Orientador

Prof. Dr. Bruno Richard Schulze
Laboratório Nacional de Computação Científica - LNCC

Prof. Dr. João Bosco Mangueira Sobral
Departamento de Informática e Estatística - UFSC

Profa. Dra. Carla Merkle Westphall
Departamento de Informática e Estatística - UFSC

Dedico esta dissertação a meus pais e demais familiares,
cujo exemplo de honestidade e trabalho tem sido
um norteador para a minha vida, e para minha
noiva, Roberta, que através de seu amor e
compreensão incondicional, me deu apoio nos
momentos mais difíceis.

Agradecimentos

Meus sinceros agradecimentos ao professor Carlos Becker Westphall, pela forma dedicada e paciente com que me orientou, e pela confiança depositada em mim desde a graduação. Serei eternamente grato ao senhor!

Agradeço aos meus amigos do NPD, pelo conhecimento que adquiri trabalhando com vocês, pela companhia nos almoços, pelos churrascos e todos os bons momentos.

Aos companheiros do LRG, pela companhia e conselhos ao longo desse período.

A todos meus amigos, os mais distantes e os mais próximos, por terem feito de mim uma pessoa melhor ao longo desses anos. Seria inútil citar todas estas pessoas, porque cometeria o equívoco de esquecer alguém.

Agradeço aos meus pais e minha irmã, por sempre terem me incentivado a seguir em frente, mesmo nos momentos em que as coisas pareciam difíceis. Devo muito a vocês!

Aos familiares da minha noiva Roberta, pelo tratamento oferecido e carinho que tiveram por mim durante esta caminhada desde a graduação. Muito obrigado mesmo!

"Comece fazendo o que é necessário,
depois o que é possível,
e de repente você estará fazendo o impossível."
São Francisco de Assis

Resumo

As tecnologias de grades computacionais são largamente utilizadas como uma forma barata de agregar poder computacional. Estas estruturas ajudam a oferecer serviços, unindo equipamentos e compartilhando os recursos como se a rede fosse um único computador. Porém, a complexidade de gerenciamento aumenta à medida que a quantidade de recursos inseridos na grade cresce. Diante do exposto, a gerência manual dos ambientes de grades é inviável. Este ambiente necessita de métodos de gerência autonômica para oferecer disponibilidade, qualidade de serviço e configurações otimizadas. Este trabalho apresenta um sistema de gerência para ambientes de grade computacional baseada em elementos autonômicos, que visa fornecer ao sistema características de auto-gerenciamento. Também no escopo deste trabalho, são descritos detalhes da implementação do sistema proposto e a realização de estudos de caso em diferentes cenários.

Palavras chave: sistemas distribuídos, grades computacionais, sistemas autonômicos, gerenciamento de redes

Abstract

Grid computing technologies are being applied as an affordable method to cluster computational power together. These structures aim to support service applications by grouping devices and shared resources in one large computational unit. However, the management complexity grows proportionally to the number of resources being integrated. From a given point up, manual management of large grid structures is unfeasible. This scenario calls for automated management methods to support availability, quality of service and optimized configurations. This work presents the design of a grid computing management system based on autonomic elements, which aims to promote characteristics of self-management. In addition, it introduces a proof-of-concept implementation and case study scenarios.

Keywords: distributed systems, grid computing, autonomic systems, network management

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Nas duas primeiras décadas da história da computação, programas e dados eram mantidos em repositórios centralizados, ocasionando gargalos tanto de performance quanto de disponibilidade para acessos à informações remotas. Com o objetivo de solucionar o problema, foi proposto o paradigma de computação distribuída[1]. Desde os anos 90, o modelo cliente/servidor e os bancos de dados distribuídos são usados para trocar informações remotas entre *hosts*, utilizando este paradigma. No princípio, os sistemas de computação distribuída consistiam em computadores conectados uns aos outros e distribuídos geograficamente em múltiplos repositórios. Deste paradigma surgiram outros conceitos atuais como computação em pares[2] (P2P), agentes móveis[3] e computação em grades[4].

Com o advento e evolução das redes de computadores e da Internet, principalmente depois da "explosão"da Web, os sistemas ficaram mais complexos devido principalmente à escalabilidade, à disponibilidade e à dinamicidade, que os serviços oferecidos devem possuir. Com a criação e a expansão da computação pervasiva, os ambientes computacionais tendem a ficar cada vez mais complexos, pois surgem no mercado continuamente uma grande variedade de minúsculos equipamentos que devem ser acoplados nos já caóticos sistemas atuais. De acordo com Mark Weiser, criador do termo computação ubíqua, que significa "em toda a parte", em seu trabalho (WEISER, 1991), percebeu a predominância cada vez maior destes pequenos dis-

[1]Segundo (COULOURIS; DOLLIMORE; KINDBERG, 2005), um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens. Isso leva as seguintes características: concorrência de componentes, falta de um relógio global e falhas de componentes independentes. Exemplos de sistemas distribuídos são: a internet, uma intranet e computação ubíqua (ou pervasiva).

[2]Segundo (COULOURIS; DOLLIMORE; KINDBERG, 2005), os sistemas *peer-to-peer* representam um paradigma para a construção de sistemas e de aplicativos distribuídos no qual dados e recursos computacionais são provenientes da colaboração de muitos *hosts* na Internet de maneira uniforme. Seu aparecimento é uma conseqüência do crescimento muito rápido da internet.

[3]Segundo (COULOURIS; DOLLIMORE; KINDBERG, 2005), um agente móvel é um programa em execução (inclui código e dados) que passa de um computador para outro em um ambiente de rede, realizando uma tarefa em nome de alguém, como uma coleta de informações, e finalmente retornando uma tarefa em nome de alguém.

[4]De acordo com (DANTAS, 2005), uma configuração de grades computacionais pode ser entendida como uma plataforma heterogênea de computadores geograficamente dispersos, no qual os usuários fazem acesso ao ambiente através de uma interface única.

positivos de computação em nosso dia-a-dia e para ele, com o passar do tempo, cada pessoa no mundo usaria muitos computadores. Com o passar dos anos, viu-se que Weiser tinha razão e os vários dispositivos da computação pervasiva estão cada vez mais presentes.

Para muitos, a computação pervasiva (*pervasive computing*) é considerada o novo paradigma do século XXI, que visa fornecer uma computação onde se deseja, quando se deseja, o que se deseja e como se deseja, através da virtualização de informações, serviços e aplicações. Este ambiente computacional consiste de uma grande variedade de dispositivos de diversos tipos, móveis ou fixos, aplicações e serviços inter-conectados.

Uma das tecnologias para suportar este novo ambiente computacional pode ser a computação distribuída em larga escala, objeto foco da computação em grade (*grid computing*) (FOSTER; TUECKE, 2001). Aplicações computacionais de processamento intensivo, executadas em uma infra-estrutura de grade, estão requerendo o uso coordenado e compartilhado de recursos em larga escala oferecidos por diferentes organizações. Como a forma como esses recursos são disponibilizados é dinâmica, as aplicações que os utilizam necessitam ser construídas de forma distribuída e adaptativa ao contexto (recursos e serviços) correntemente disponível.

A junção de paradigmas como rede de sensores[5] e computação pervasiva às grades de computadores tornam-na um ambiente claramente heterogêneo com diversos dispositivos, ambientes e infra-estruturas diferentes. Devido às características citadas, esse ambiente torna-se extremamente complexo e de gerência, freqüentemente, ineficaz, principalmente quando feito por humanos.

Segundo (IBM, 2005a), tradicionalmente, redes e sistemas de gerenciamento são processos controlados manualmente, levando um ou mais operadores humanos a gerenciar todos os aspectos dos sistemas de computação. Neste ambiente, o operador é fortemente integrado no processo de gerenciamento e a sua tarefa é executar comandos de sistema de baixo-nível para resolver problemas iminentes. Embora esta forma de gerenciamento, que mantém o ser humano dentro do sistema, fosse apropriada no passado, torna-se cada vez mais inadequada em sistemas modernos. As organizações de tecnologia da informação (TI) encontram cada vez mais dificuldades em manter e gerenciar sistemas distribuídos em larga escala (como as grades computacionais), porque estes necessitam estar sempre ativos e disponíveis.

Segundo a pesquisa (PATTERSON et al., 2002), realizada pela *University of California*, uma parcela considerável do orçamento das organizações de TI é gasto com prevenção e recu-

[5]De acordo com (ESTRIN et al., 2001), a *Wireless Sensor Network* - WSN é uma forma de integrar sensores (acústicos, infravermelho, câmera, temperatura, calor, sísmico, etc), comunicação e fonte de alimentação em dispositivos miniaturizados usando apenas tecnologia de circuitos digitais, comunicação sem fio e sistemas microeletromecânicos, de maneira a fundar uma tecnologia na área de redes ad hoc.

peração de sistemas depois de uma pane.

Em outubro de 2001, o Dr. Paul Horn da IBM introduziu o conceito de sistemas autonômicos (HORN, 2001) como uma forma de tratar o crescimento da complexidade de administração dos modernos sistemas computacionais. Segundo este trabalho, o custo com mão-de-obra das empresas de TI chega a 70% dos gastos totais e o custo continua subindo. Assim, para Horn, a indústria de TI está a beira de uma crise, não de hardware, mas de mão-de-obra especializada para suportar a gerência dos complexos ambientes computacionais.

A computação autonômica é a solução que propõe realocar muitos dos administradores responsáveis pela gerência do sistema, tirando-os da execução de tarefas de baixo nível e usando sua experiência na definição de políticas de alto nível. A visão da computação autonômica é melhorar a gerência dos sistemas computacionais, introduzindo auto-gerenciamento (*self-management*) (HINCHEY; STERRITT, 2006) para configuração, proteção, otimização e regeneração.

1.1 Definição do problema e proposta

Como relatado no manifesto (HORN, 2001) feito pela IBM, o principal obstáculo para o progresso da indústria de TI é a crise de complexidade nos sistemas computacionais. A companhia cita ambientes e aplicações que possuem dez milhões de linhas de código e requerem profissionais muito especializados para instalar, configurar, atualizar e manter os sistemas.

A necessidade de interligar vários ambientes heterogêneos, um dos principais requisitos das grades computacionais, introduz novos níveis de complexidade. Atualmente, as grades computacionais são reconhecidas como um ambiente computacional dinâmico e heterogêneo capaz de suportar computação distribuída em larga escala. Apesar de ser um ambiente intrinsecamente complexo, a configuração e a gerência é realizada pelo esforço humano, ficando sujeita à lentidão para tomada de decisão ou até mesmo, erros por parte dos administradores. A fim de tratar este problema, necessita-se de uma solução na qual o trabalho de gerência não fique a cargo somente de administradores humanos.

Observando este cenário, surge uma pergunta importante: *como fazer com que um ambiente bastante heterogêneo e de grande complexidade como as grades computacionais, não fique sujeito à gerência manual, que muitas vezes é ineficiente?*

Visando responder esta pergunta, este trabalho propõe uma sistema de gerência autonômica de grades computacionais, baseada em uma base teórica sólida, com suporte ao para-

digma de computação autonômica (HARIRI et al., 2006), fornecendo propriedades de auto-gerenciamento, na qual a função do operador humano é redefinida, utilizando sua experiência na definição de objetivos gerais e políticas para o controle do sistema, ao invés de envolvê-lo em um processo decisório com uma visão interativa e fortemente acoplada.

1.2 Organização do trabalho

O trabalho está organizado da seguinte forma:

- **Capítulo 1 - Introdução**: apresenta a introdução ao tema, contexto e problema da pesquisa, colocando suas questões e objetivos, bem como, a definição da proposta e a organização do estudo;
- **Capítulo 2 - Computação Autonômica**: discorre sobre conceitos, características e peculiaridades da computação autonômica. Detalha também o conceito de auto-gerenciamento, fundamental para o trabalho;
- **Capítulo 3 - Grades Computacionais**: apresenta conceitos fundamentais sobre o ambiente na qual o sistema proposto é utilizado. Além disto, discorre sobre alguns *middlewares* existentes, entre eles o Grid-M, utilizado no estudo de caso;
- **Capítulo 4 - Sistema para Gerência Autonômica de Grades Computacionais**: discorre, em essência, sobre a arquitetura proposta, características e componentes da mesma, bem como sobre os algoritmos de roteamento implementados. Explica o motivo e a implementação das soluções adotadas. É criado um estudo de caso, usando o Grid-M, e testes são feitos com o intuito de verificar a viabilidade da proposta;
- **Capítulo 5 - Conclusão**: apresenta as considerações finais, descrevendo os objetivos alcançados, as limitações da pesquisa e as novas direções de estudo em função do complemento do problema exposto.

2 *Computação Autonômica*

2.1 Definições e características

Quando o vice-presidente de pesquisa da IBM, Paul Horn, introduziu a idéia de computação autonômica na *Nation Academy of Engineers* da Universidade de Harvard, ele escolheu um nome deliberadamente com uma conotação biológica.

O termo *Autonomic Computing* vem da tentativa de fazer com que um sistema de computação tenha funções semelhantes às desempenhadas pelo sistema nervoso humano (IBM, 2005a). Quando uma pessoa corre, ou faz alguma atividade física, o seu organismo nota a mudança e efetua modificações que permitem alcançar seus objetivos. Particularmente, o coração começa a bater mais rápido, a respiração acelera e a pele começa a suar, para começar a gastar o calor em excesso. Todas as mudanças do organismo acontecem de modo autônomo, sem que a pessoa necessite pedir para que estas mudanças aconteçam. Como as mudanças ocorrem automaticamente, a pessoa pode ocupar seu tempo com outras atividades.

Um sistema autonômico é um sistema capaz de regular seus próprios parâmetros funcionais sem atrapalhar os objetivos principais do sistema. Esse tipo de sistema é capaz de mostrar um perfeito funcionamento também em condições de *stress* particulares ou de carga excessiva de trabalho. As vantagens de possuir uma gestão autonômica dos recursos e da configuração são notáveis e extremamente importantes. Essas são ainda mais evidentes em um ambiente heterogêneo e fortemente dinâmico como o ambiente de grades computacionais. Um ambiente autonômico consegue ter um melhor nível de otimização e então, desfrutar melhor os seus recursos.

De acordo com (IBM, 2005a), a eficiência dos processos de TI é medida utilizando métricas, como tempo transcorrido até completar o processamento, percentual de processos executados corretamente e custos para executar o processo. Sistemas auto-gerenciáveis podem afetar positivamente estas métricas, ajudando a melhorar a responsabilidade, a qualidade de serviço e o tempo de resposta.

A jornada em direção à completa autonomia ainda levará alguns anos, mas existem muitas etapas importantes nesse longo caminho. Primeiramente, funções autonômicas irão coletar e agregar informações para ajudar nas decisões tomadas por administradores humanos. Além disso, servirá como guia, sugerindo ações e caminhos a serem seguidos.

2.2 Níveis evolutivos

Devido à dificuldade de implementar estas características listadas em um só passo, são definidos 5 níveis de evolução:

- Nível 1 (básico): é o estado da arte atual do gerenciamento de redes, no qual o gerente humano gerencia os componentes manualmente;
- Nível 2-4: definem estágios intermediários dos sistemas de TI em direção a sistemas completamente autonômicos;
- Nível 5: neste nível tem-se um sistema totalmente autonômico.

2.3 Auto-gerenciamento (*Self-management*)

A principal característica dos sistemas de computação autonômicos é a capacidade de auto-gerenciamento, liberando os administradores dos detalhes de operação e manutenção, além de prover aos usuários um sistema que trabalhasse em desempenho otimizado durante as 24 horas do dia nos 7 dias da semana (KEPHART; CHESS, 2003).

Como o nome diz, os sistemas autonômicos irão manter e ajustar suas operações de acordo com a mudança dos componentes, fluxos de trabalhos, demandas e condições externas, além de se ater as falhas de hardware ou software, tanto maliciosas quanto as sem intenção. O sistema poderia continuamente monitorar seu próprio uso e checar por atualizações de seus componentes. Um fato interessante nos sistemas de computação autonômica (*autonomic computing system* - ACS) é a capacidade de regressão. Exemplo disso é a figura 2.1, na qual (KEPHART; CHESS, 2003) demonstra como o trabalho de atualização poderia funcionar em sistema autonômico.

Na figura 2.1, a atualização introduz cinco módulos de software (azul), sendo que cada um é um elemento autonômico (*Autonomic Element* - AE). Minutos depois da instalação, testes de regressão encontraram falhas nas saídas de três módulos novos (linhas vermelhas), e o sistema

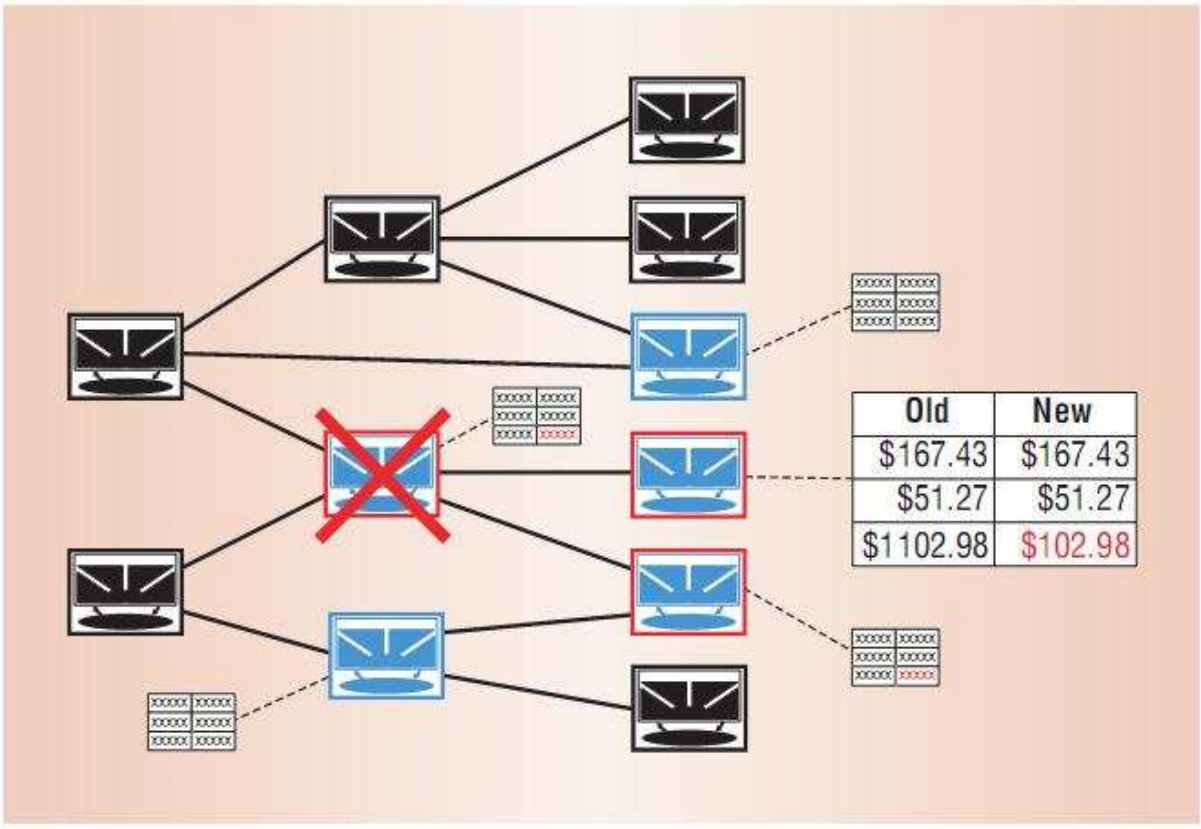


Figura 2.1: Diagnóstico de um problema de atualização (KEPHART; CHESS, 2003)

retorna imediatamente para a versão anterior. Um elemento autonômico, responsável por detectar problemas, obtém informações sobre as dependências entre os elementos (linhas entre os elementos). Levando em consideração seu conhecimento sobre as dependências, o elemento responsável por detectar problemas analisa os arquivos de *log* e infere qual dos três módulos potencialmente defeituosos é o responsável pelo problema (vermelho com o X). É gerado um relatório sobre a falha, contendo informações sobre o diagnóstico, que é enviado para o desenvolvedor de software, responsável por depurar o módulo e torná-lo disponível para futuras atualizações.

Como dito anteriormente, o corpo humano é capaz de monitorar, controlar e regular seu comportamento sem intervenção externa. Um sistema autonômico provê essas capacidades para sistemas heterogêneos, complexos e de larga escala.

De acordo com (HORN, 2001), para alcançar autonomia completa, um sistema de computação autonômico deve implementar oito áreas (auto-configuração, auto-regeneração, auto-otimização, auto-proteção, auto-conhecimento, sensibilidade ao contexto, abertura e antecipação). Essas áreas permitem obter auto-gerenciamento, fornecendo vantagens aos usuários do próprio sistema.

2.3.1 Auto-configuração (*Self-configuration*)

Um sistema deve ser capaz de configurar-se de modo otimizado sem nenhuma intervenção externa (como a humana). A configuração deve ser dinâmica e feita de acordo com políticas de alto nível (representando objetivos do nível de negócios), que especificam quais características são necessárias, não o que deve ser feito para alcançá-las. Um exemplo de auto-configuração ocorre na inserção de um componente, no qual o resto do sistema deve adaptar-se a presença desse novo elemento, assim como ocorre com uma célula do corpo ou uma pessoa em uma população. Assim que introduzido, o novo elemento irá registrar a si mesmo e suas capacidades para que os outros elementos possam usá-lo ou modificar seus próprios comportamentos. A configuração dos sistemas de informação resultam em um trabalho complexo, já que é uma operação longa, suscetível a erros. Como a configuração de um sistema é um processo necessário durante toda a vida do mesmo, a capacidade autonômica faz com que o sistema se adapte a mudanças de ambiente.

2.3.2 Auto-regeneração (*Self-healing*)

De acordo com (KEPHART; CHESS, 2003), a IBM e outras companhias de TI possuem grandes departamentos voltados à identificação e rastreamento das principais causas de falhas nos complexos sistemas computacionais. Alguns problemas levam semanas de trabalho de equipes de programadores para que sejam diagnosticados e corrigidos. Outros são ainda pior, pois aparecem repentinamente e logo após desaparecem misteriosamente, sem que pudessem deixar um diagnóstico.

A grande vantagem de utilizar a auto-regeneração é dar ao sistema a capacidade de efetuar diagnósticos sobre si mesmo sem intervenção externa, com o objetivo de encontrar eventuais erros ou sub-sistemas que não funcionam de modo correto, como ocorre na figura 2.1. O *self-healing* é fundamental para que se possa responder de modo reativo a situação de erro ou pane imprevista de parte do sistema. Uma das características desejáveis de um sistema de informação é a capacidade de funcionar também na presença de falhas limitadas. Como alguns serviços ativos no sistema são de grande importância, é necessário garantir um mecanismo que permita ao sistema responder de modo adaptativo também a problemas de dimensões modestas. O auto-diagnóstico torna possível ao sistema modificar-se para conseguir corrigir alguma falha e continuar suas atividades fundamentais.

2.3.3 Auto-otimização (*Self-optimization*)

O sistema deve ser capaz de otimizar-se em modo autonômico, permitindo sempre a melhor utilização possível dos recursos disponíveis. Segundo (KEPHART; CHESS, 2003), *middlewares* complexos, ou sistemas de banco de dados como o Oracle e o DB2, possuem centenas de parâmetros funcionais que podem ser ajustados para melhorar o sistema, mas poucas pessoas sabem como modificá-los.

A auto-otimização permite que parâmetros funcionais do sistema sejam modificados sem intervenção externa, garantindo uma resposta positiva também no caso de situações de carga de trabalho imprevistas. É desejável que um sistema seja capaz de tirar o máximo dos recursos que tenha a disposição. Para poder atingir isto, o ACS deve procurar continuamente modos de melhorar suas operações, identificando oportunidades para melhorar sua eficiência em performance ou custo, principalmente levando em consideração informações sobre a carga de trabalho que chegam dos usuários e das características funcionais dos recursos, realizando, sempre que possível, o balanceamento de carga. Assim como os músculos ficam mais fortes através de exercícios e o cérebro modifica seus circuitos durante o aprendizado, o sistema autonômico otimizará seus parâmetros durante a execução e aprenderá a fazer escolhas apropriadas (KEPHART; CHESS, 2003).

2.3.4 Auto-proteção (*Self-protecting*)

(KEPHART; CHESS, 2003) afirma que apesar da existência de *firewalls* e ferramentas de detecção de intrusão (IDS[1]/IPS[2]), até o momento, os humanos são os responsáveis por decidir como deve ser feita a proteção contra ataques maliciosos e falhas em cascata.

Para evitar essa situação, os sistemas autonômicos devem realizar auto-proteção nos dois sentidos: devem defender o sistema como um todo contra ataques maliciosos de larga escala, além de não permitir que o sistema entre em colapso com falhas em cascata.

Assim, o sistema deve ser capaz de antecipar e encontrar eventuais intrusões ou violações de alguns parâmetros do sistema, permitindo um funcionamento seguro e controlado de todas as próprias operações. Essa característica vai em direção da criação de sistemas sempre mais seguros e confiáveis. Quanto mais serviços importantes ativos o sistema executa, maior é a necessidade de garantir segurança. Também neste caso, a possibilidade de efetuar estas operações

[1] O IDS (Intrusion Detection System, ou Sistema de Detecção de Intrusos) é um sistema que monitora o tráfego e detecta se a rede está tendo acessos não autorizados.

[2] O IPS (Intrusion Prevention System, ou Sistema de Prevenção de Intrusos), além de detectar o acesso irregular, também é capaz de tratá-lo.

de modo autonômico garante diminuir notavelmente a possibilidade de falhas humanas. Além disso, o monitoramento ativo dos parâmetros de segurança permite individualizar eventuais falhas ou problemas no sistema e corrigi-los, antes que sejam capazes de criar falhas no sistema de proteção e resistência.

2.3.5 Auto-conhecimento (*Self-awareness ou Self-knowledged*)

Um ACS deve ter a habilidade de conhecer a si mesmo. Deve ser sensível a mudanças dos seus componentes, saber sobre o estado e a disponibilidade dos recursos. Deve saber também quais recursos estão lentos e quais podem ser compartilhados.

2.3.6 Sensibilidade ao contexto (*Context-awareness*)

Um sistema autonômico deve conhecer seu ambiente e o contexto que cerca suas atividades, agindo de acordo com o mesmo. Assim, será sensível às mudanças no ambiente e reagirá a estas com a adoção de novas políticas, as quais estarão em perfeito acordo com os sistemas vizinhos, ou seja, uma grande capacidade de adaptação.

2.3.7 Abertura (*Openness*)

Um ACS necessita ser altamente portável, operando sobre múltiplas plataformas e com uma grande variedade de componentes. O conceito base dessa característica é a heterogeidade. Mesmo que independente para gerenciar a si próprio, deve se preocupar em funcionar em um mundo heterogêneo e implementar padrões abertos, ou seja, um sistema autonômico não deve ser uma solução proprietária.

2.3.8 Antecipação (*Anticipatory*)

Um ACS deve antecipar a otimização dos recursos, enquanto esconde a complexidade dos usuários finais e deve atender satisfatoriamente as requisições dos usuários. Além disso, deve gerenciar os recursos de TI de modo a diminuir a distância entre os objetivos de negócios e os objetivos pessoais.

Os primeiros sistemas autonômicos tratavam essas áreas como distintas, com diferentes equipes de desenvolvimento criando soluções para cada uma delas. Ultimamente, surgem arquiteturas de cunho geral (como deste trabalho), que visam tratar todas as características do auto-gerenciamento de uma só vez.

2.4 Arquiteturas de Computação Autonômica

2.4.1 Elementos Autonômicos (*Autonomic Element* - AE)

Os elementos autonômicos são as unidades funcionais dos sistemas autonômicos, como os tijolos de uma construção, que contém recursos e disponibilizam serviços para usuários e outros AEs. Eles gerenciam seus comportamentos internos e suas relações com outros elementos do sistema, de acordo com políticas estabelecidas por humanos ou por outros AEs. O comportamento autonômico de todo o sistema surge das inúmeras interações entre os elementos autonômicos. Segundo (KEPHART; CHESS, 2003), infra-estruturas distribuídas e orientadas a serviço como as grandes computacionais são ideais para suportar elementos autonômicos e suas interações.

Essencialmente, um elemento autonômico consiste de um sistema de controle fechado (figura 2.2). Teoricamente, sistemas que apresentam laços de controle podem controlar um sistema sem intervenção externa e podem mantê-lo em um estado específico. O conceito é fundamental para o ACS porque introduz a desejada autonomia.

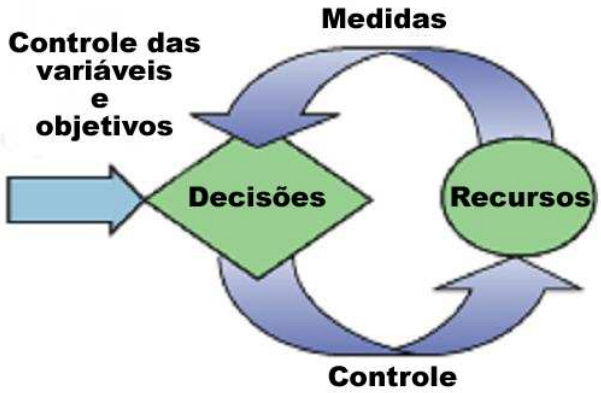


Figura 2.2: Laço de controle de um AE (HERRMANN; MüHL; GEIHS, 2005)

Porém, segundo (HERRMANN; MüHL; GEIHS, 2005), o grande desafio é construir laços de controles eficientes. A idéia de laços de controle é bem conhecida de outras aplicações na área da engenharia. Exemplos disso são o termostato, que consiste de um sensor de temperatura e de um par de válvulas para controle de fluxo, e o sistema de freio anti-derrapagem (ABS[3]). Laços de controle podem gerenciar parâmetros de sistema com base em um conjunto de pontos pré-definidos e observar constantemente a mudança dos mesmos. Assim, para o termostato, uma pessoa define um conjunto de valores (temperaturas desejadas), e o termostato mede a temperatura e reage controlando o fluxo de calor (usando as válvulas). Já o ABS detecta se a roda

[3]Anti-lock Breaking System

está travada e reage reduzindo o poder de frenagem até que a mesma volte a girar normalmente. Para (HERRMANN; MüHL; GEIHS, 2005), o fato de companhias automobilísticas gastarem considerável recurso desenvolvendo e calibrando seus sistemas de ABS mostra que esse é um ambiente complexo, não trivial, que possue numerosas influências externas.

Todavia, os modernos sistemas distribuídos como as grades computacionais são algumas ordens de magnitude mais complexas que os dois exemplos. Em um sistema ABS, todos os estados e influências externas são conhecidas previamente. Normalmente, não é o que ocorre com sistemas computacionais, porque estes apresentam uma grande variedade de componentes heterogêneos.

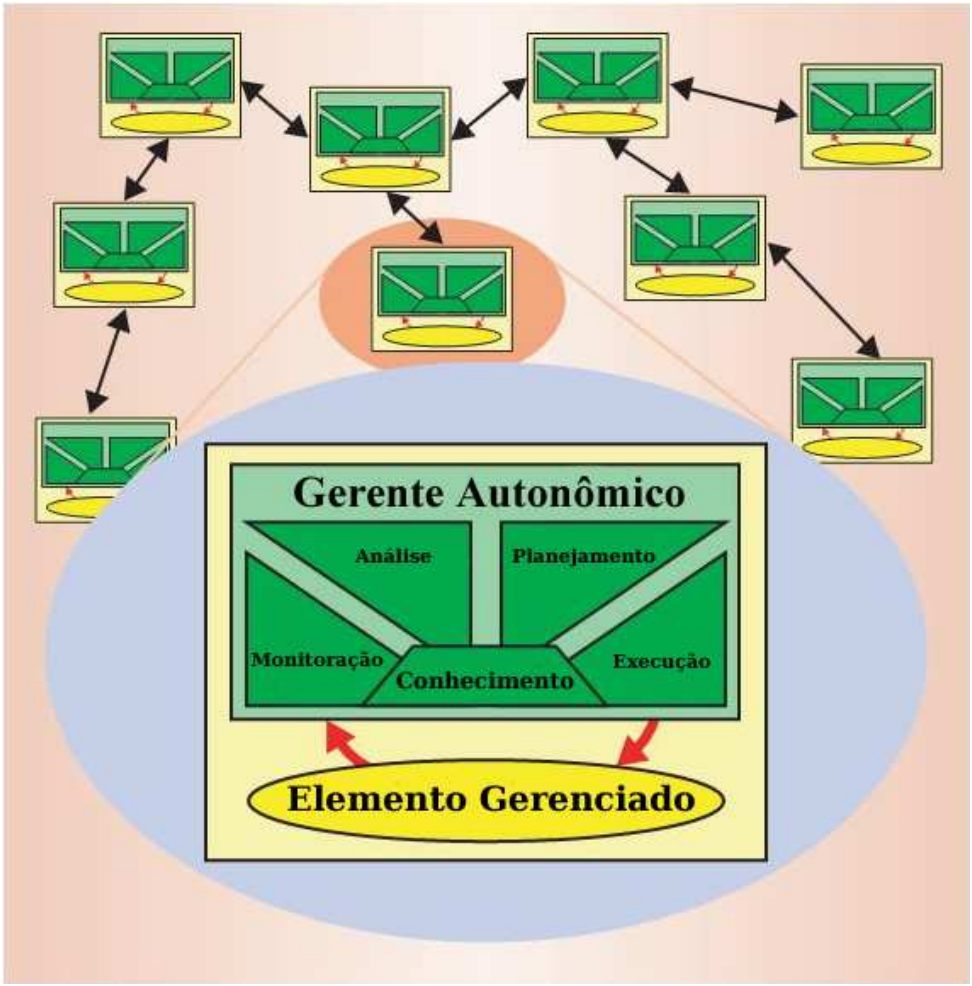


Figura 2.3: Arquitetura de um Elemento Autonômico (KEPHART; CHESS, 2003)

Um elemento autonômico consiste de um ou mais elementos gerenciados, acoplados a um único gerente autonômico (*autonomic manager* - AM), que controla os elementos gerenciados (*managed element* - ME). A figura 2.3 mostra exatamente isto. Os elementos gerenciados podem ser um recurso de hardware, como uma impressora ou um processador, ou um de software, como um banco de dados ou um serviço de diretório. Para (KEPHART; CHESS, 2003), ele-

mentos gerenciados são essencialmente equivalentes aos recursos e serviços que se encontram em sistemas não autonômicos, embora estejam aptos para disponibilizar dados ao gerente autonômico que irá monitorá-los e controlá-los. De acordo com (GANEK; CORBI, 2003), os elementos gerenciados devem incluir sensores e atuadores. Os sensores tem a função de capturar e enviar as informações sobre o estado atual do elemento gerenciado, que posteriormente são comparados com as regras da base de conhecimento do Elemento Autonômico. As ações sobre os elementos gerenciados são feitas pelos atuadores.

O que difere um sistema autonômico de um não autonômico é a presença do gerente autonômico. Através do monitoramento dos elementos gerenciados e de seu ambiente externo, o gerente autonômico é capaz de construir e executar planos de execução baseados na análise das informações enviadas, tirando de administradores humanos a responsabilidade de gerenciá-los.

2.4.2 Blueprint IBM - Arquitetura genérica

A IBM foi uma das precursoras da computação autonômica. Com o trabalho (IBM, 2005a), a empresa introduziu uma arquitetura genérica para a construção de sistemas autonômicos (figura 2.4) seguida por um grande número de trabalhos na área.

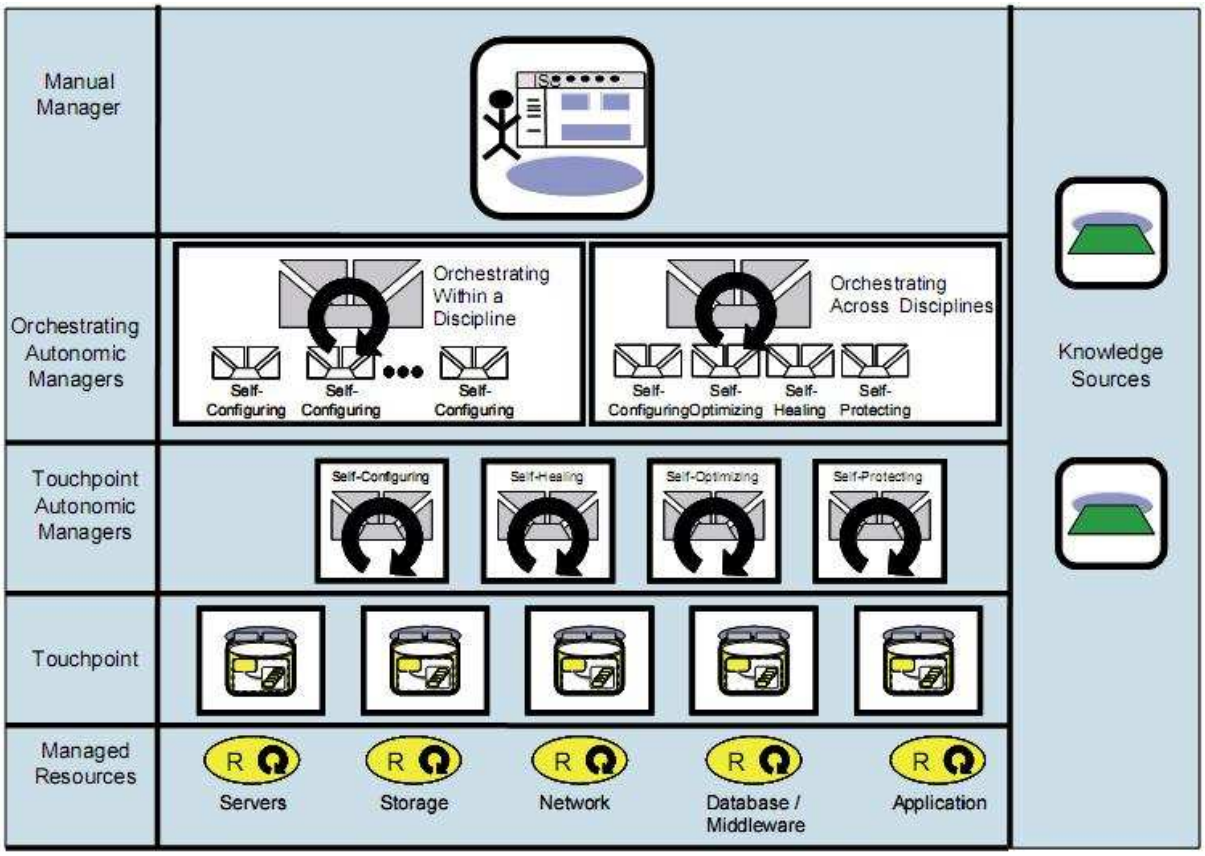


Figura 2.4: Arquitetura Blueprint IBM (IBM, 2005a)

A figura 2.4 mostra uma topologia para a composição de blocos que garante um sistema autonômico. A camada inferior contém os componentes do sistema, ou recursos gerenciados, que compõe a estrutura de TI. Como dito na seção anterior, esses recursos gerenciados podem ser tanto recursos de hardware como serviços da grade. A camada seguinte incorpora interfaces de gerenciamento consistentes e padronizadas para acesso e controle dos recursos gerenciados. Estas interfaces padronizadas são entregues pelo *manageability endpoint*. As camadas 3 e 4 automatizam uma parte dos processos de TI, usando um gerente autonômico (*autonomic manager*).

Na camada 3 (figura 2.4), existe um elemento gerente que implementa cada uma das categorias do auto-gerenciamento (auto-configuração, auto-regeneração, auto-otimização e auto-proteção), discutidos em detalhes na seção 2.3. A camada 4 possibilita que gerentes autonômicos possam ser orquestrados por outros gerentes autonômicos. A camada 5 (topo) ilustra o gerenciamento manual que provê uma interface de gerenciamento comum, para que os profissionais de TI possam usar a console para solucionar problemas.

Para um componente ser auto-gerenciável, é necessário que o mesmo possua métodos de coleta autonômicos, meios para analisar estas informações para determinar se algo deva ser mudado. Além disso, deve possuir modos de criar planos, ou seqüencias de ações, que especificam as mudanças necessárias, e deve estar apto para executar estes planos. Quando estas funções podem ser automatizadas, um inteligente laço de controle é formado (IBM, 2005a). Como dito na seção 2.4.1, o conceito de laço de controle fechado é fundamental para implementar um sistema autonômico, pois é ele que possibilita o comportamento autonômico. Como é o gerente autonômico o componente que implementa o laço de controle, ele é o responsável pela autonomia do sistema.

2.5 Gerente Autonômico

O gerente autonômico é composto de quatro partes: monitoramento, análise, planejamento e execução, como mostra a figura 2.5. Além destes, o AM possui também uma fonte de conhecimento, importantíssima para um sistema autonômico. (IBM, 2005a) fornece vários detalhes sobre cada uma destas partes. Devido a sua importância, será dedicado a seção 2.5 para considerações sobre o gerente.

A arquitetura sugerida pela IBM não foi criada para trabalhar com ambiente de grade. Porém, essa arquitetura é a precursora na área e apresenta características fundamentais para fornecer autonomia a qualquer sistema computacional. Por isso, é considerada a base de arquitetura

proposta neste trabalho.

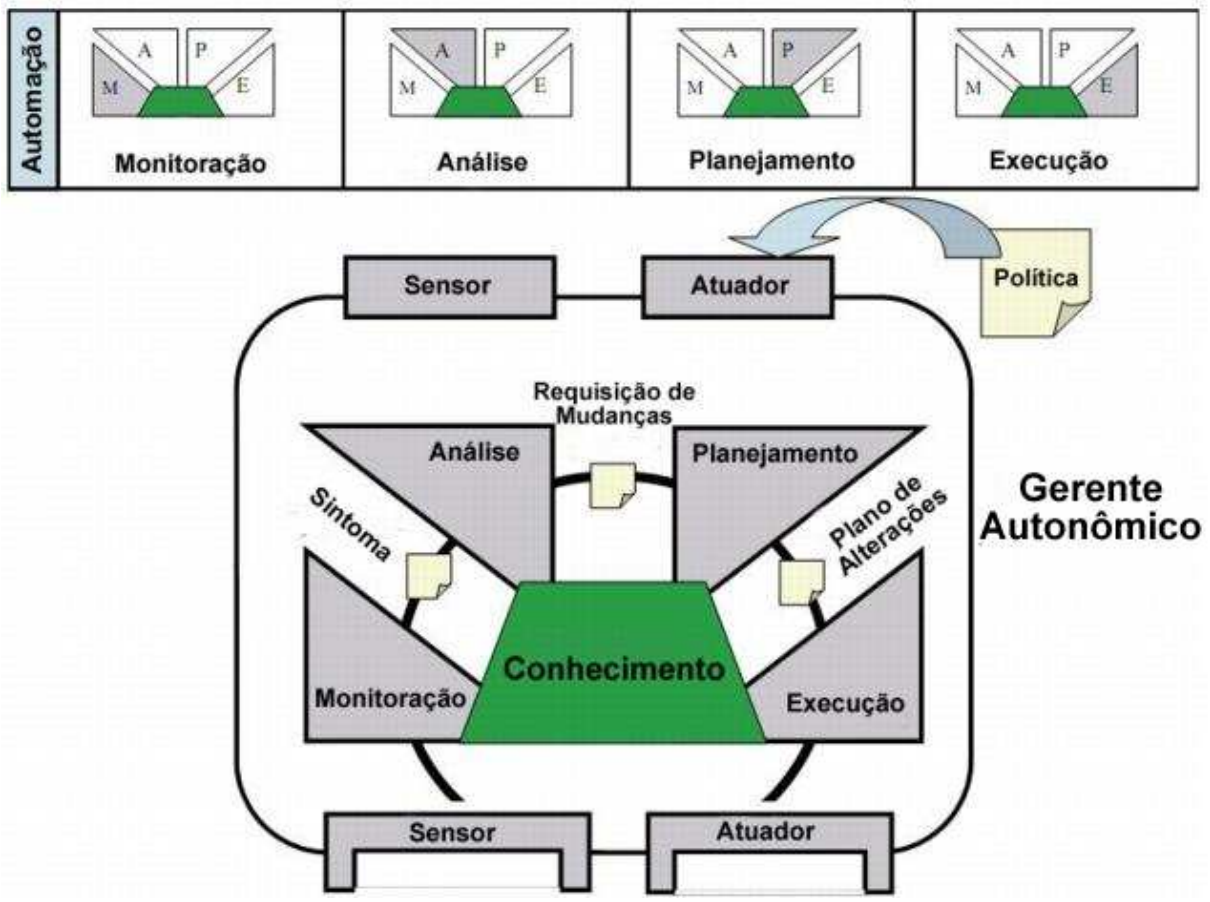


Figura 2.5: Detalhes do Gerente Autonômico (*Autonomic Manager*) (IBM, 2005a)

2.5.1 Monitoramento

O Monitoramento coleta detalhes dos recursos gerenciados e os relaciona com os sintomas conhecidos. Os detalhes podem ser informações sobre a topologia, métricas (estado, capacidade oferecida, *throughput*, etc) e propriedades de configuração. Alguns destes dados são estáticos ou trocam raramente, outros entretanto são dinâmicos, trocando valores continuamente através do tempo. As funções de monitoramento agregam, relacionam e filtram estes detalhes até determinar o sintoma que necessita ser analisado. Por exemplo, o Monitoramento poderia agregar e relacionar o conteúdo dos eventos recebidos de múltiplos recursos para determinar um sintoma que relata uma combinação destes eventos. Logicamente, o sintoma é passado para a função de análise. A habilidade do gerente autonômico de organizar-se com rapidez é crucial para o seu sucesso organizacional (IBM, 2005a).

2.5.2 Análise

A Análise provê mecanismos para observar e analisar situações, determinando se alguma troca deve ser feita. Por exemplo, o requisito para efetivar uma troca pode ocorrer quando o analisador determinar que alguma política não está sendo cumprida. A Análise modela complexos comportamentos, implementados por técnicas que permitem ao sistema aprender sobre o ambiente e ajudar a prever futuras tendências. Os gerentes autonômicos estão aptos a executar complexas análises de dados e tratar os sintomas levantados pela área de monitoramento. Quando uma alteração é necessária, as funções de análise geram uma requisição de mudanças, que é passada para a área de planejamento. A requisição descreve as modificações necessárias, de acordo com o analisador (IBM, 2005a).

2.5.3 Planejamento

O Planejamento cria ou seleciona procedimentos para realizar alterações desejadas em recursos gerenciados. As funções desta área geram um plano apropriado de mudanças, que representam o conjunto das alterações consideradas necessárias. Assim que uma decisão é tomada, um plano de alterações é passado para a área de execução (IBM, 2005a).

2.5.4 Execução

A área de execução provê mecanismos para agendar e executar alterações necessárias para o sistema. Uma vez que o gerente autonômico gerou o plano de alterações que corresponde à requisição de mudanças, algumas ações são necessárias para modificar o estado de um ou mais recursos gerenciados. A Execução é responsável por cuidar para que os procedimentos gerados pelo Planejamento sejam executados. Por exemplo, uma parte da execução de um plano de alterações poderia envolver alterações na fonte de conhecimento (IBM, 2005a).

2.5.5 Fonte de conhecimento

A fonte de conhecimento é a implementação de um registro, dicionário, base de dados ou outros repositórios que fornecem acesso ao conhecimento. Em um ACS, o conhecimento consiste de tipos de dados com sintaxe e semântica próprias, como sintomas, políticas, requisição de mudanças e plano de alterações. Este conhecimento pode ser armazenado em uma fonte de conhecimento, podendo ser usada para estender as capacidades conhecidas do gerente autonômico. Um gerente autonômico pode carregar conhecimento de uma ou mais fontes, permitindo

que o mesmo execute tarefas de gerência adicional, como reconhecimento particular de sintomas.

Estas quatro partes (resumidas pela tabela 2.4) com os sensores e atuadores provém o *loop* fechado que permite criar o *self-management*.

2.5.6 Arquiteturas de software para computação autonômica

McCann (MCCANN; HUEBSCHER, 2004) em seu trabalho identificou duas categorias de abordagens para os sistemas autonômicos: fracamente e fortemente acoplada. Porém, as duas abordagens tem conceitos em comum e em alguns casos é difícil enquadrar um projeto em uma categoria particular. Exemplos de ambas as abordagens serão dados nas tabelas 2.1 e 2.2.

Abordagem fortemente acoplada

A abordagem fortemente acoplada é construída usando agentes inteligentes, formando um sistema multi-agentes (WOOLDRIDGE, 2002). Nos sistemas autonômicos complexos cada agente tem seus próprios objetivos, guiando suas próprias ações. Um agente é pró-ativo e possui responsabilidade social. Segundo (KEPHART; CHESS, 2003), essa última característica pode fazer com que instabilidades sejam causadas em todo o sistema devido a uma mudança de comportamento de um agente. Além disto, deve-se também levar em consideração a dificuldade de definir objetivos individuais dos agentes para alcançar os objetivos do sistema. Em sistemas autonômicos, deve-se prover meios para que os objetivos possam ser expressos em notação de alto nível e confiar que os agentes entenderão esta notação, conseguindo determinar automaticamente o que é necessário fazer para alcançar os objetivos traçados.

Se comparada com a arquitetura fracamente acoplada, os sistemas multi-agentes apresentam um característica inerentemente distribuída. Assim, como não existe uma infra-estrutura centralizadora, os agentes devem monitorar a si mesmo (monitoramento interno) e também outros agentes (monitoramento externo). Para que se possa realizar o monitoramento externo, (STERRITT, 2002) propõe uma abordagem na qual cada agente envia uma mensagem regularmente em canal compartilhado para que os outros agentes possam escutar. Esta mensagem provê um resumo do estado de um agente autonômico para outros agentes interessados. De acordo com (MCCANN; HUEBSCHER, 2004), esta abordagem foi utilizada com sucesso na *Open Grid Services Architecture* (OGSA) (GLOBUS-OGSA,) e pela NASA na missão Deep Space 1 (DS1) (WYATT, 1998). Em ambos os casos, a mensagem enviada fornece limitadas informações sobre o componente gerenciado.

Tabela 2.1: Sumários de trabalhos com abordagem fortemente acoplada

Trabalho	**Domínios**	**Características**
(CHAO; JAMES, 2001)	Framework para sistemas multi-agentes	Middleware baseado no CORBA para monitoramento e cooperação
(STERRITT, 2002)	Componentes autonômicos	*Heartbeat* ou pulso para monitoramento
(GEORGIADIS; MAGEE, 2002)	Arquitetura para componentes auto-organizáveis	Componentes auto-organizáveis que visam organizar o sistema como um todo
(KUMAR; COHEN, 2000)	Arquitetura adaptativa baseada em agentes	Arquitetura que usa um agente intermediador para prover tolerância a falta
(BIGUS et al., 2002)	Ferramenta ABLE	Ferramenta para construção de sistemas multi-agentes

Exemplos de trabalhos com esta abordagem são mostrados na tabela 2.1.

Abordagem fracamente acoplada

Nesta abordagem, os componentes individuais não são por si só autonômicos. Existe uma infra-estrutura autonômica que é separada do ambiente de execução e trata as informações de todos os componentes do sistema. Assim, essa infra-estrutura executa em uma máquina separada e por isso, esta abordagem é chamada fracamente acoplada.

O sistema em execução é monitorado através de *probes*, através da qual seus dados são coletados e enviados para o gerente. Assim, o gerente pode sugerir atualizações, adaptações ou reparos do sistema, baseadas nas informações enviadas. Essas são analisadas e um plano de execução é traçado. Após o plano feito, ele pode ser executado. (MCCANN; HUEBSCHER, 2004) citam como vantagem desta abordagem em relação à outra, a completa separação entre o comportamento autonômico e o ambiente de execução, podendo adaptar-se facilmente em sistemas pré-existentes. A única intervenção direta com o sistema é através das *probes* que o monitoram, e atuadores que fazem ajustes e reconfigurações.

Devido à natureza centralizada desta abordagem, podem ocorrer problemas relacionados a sistemas centralizados, como a falha da máquina que executa o processamento das informações. Portanto, é essencial o cuidado com a tolerância a faltas nesta infra-estrutura.

Um sistema que usa este tipo de abordagem é o proposto por (VALETTO; KAISER, 2002), que usa um fluxo de trabalho descentralizado, usando agentes que são parte de uma infra-estrutura adaptativa separada do ambiente de execução.

Tabela 2.2: Sumários de trabalhos com abordagem fracamente acoplada

Trabalho	**Domínios**	**Características**
(GARLAN; SCHMERL, 2002)	Arquitetura adaptável para sistemas autonômicos	Fornece *probes* para monitoramento do sistema em tempo de execução, dando ao sistema um comportamento adaptável
(LEMOS; FIADEIRO, 2002)	Arquitetura para tolerância a faltas	Componentes considerados como caixa preta
(DASHOFY; HOEK, 2002)	*Framework* para sistemas adaptáveis	Fornece uma linguagem de descrição (xADL 2.0) para estruturar a arquitetura
(VALETTO; KAISER, 2002)	Adição de comportamento autonômico a sistemas existentes	Comportamento autonômico fornecida por uma infra-estrutura multi-agentes chamadas *Workflakes*

Exemplos de trabalhos com esta abordagem são mostrados na tabela 2.2.

2.6 Agentes e sistema multi-agentes

Segundo alguns pesquisadores como (TESAURO et al., 2004), (DIAO et al., 2003), (WOLF; HOLVOET, 2003), a forma mais adequada de se alcançar um sistema autonômico é através dos sistemas multi-agentes (*multi-agent system* - MAS). Estes sistemas utilizam a arquitetura fortemente acoplada, ideal para ambientes distribuídos como as grades computacionais.

2.6.1 Definições

Definir o que é um agente é um pouco complicado, devido ao termo ser utilizado em diversas áreas. Uma definição bastante aceita é: um agente é uma entidade que percebe o mundo através de seus sensores e atua sobre ele por meio dos atuadores. Assim, nos agentes de *software*, os sensores seriam as entradas e os atuadores as saídas do sistema ou forma pela qual o agente age sobre o meio.

Alguns autores vêem os agentes de forma isolada, enquanto outros preferem tratá-los como entidades que atuam de maneira colaborativa, formando o conceito de sistemas multi-agentes.

2.6.2 Características dos agentes de software

Para (BRENNER; WITTIG; ZARNEKOW, 1998), um agente deve possuir algumas características. Não necessariamente deve ter todas elas, mas grande parte. As características desejáveis dos agente de software são:

- **Autonomia**: os agentes devem perseguir seus objetivos de modo autonômico, ou seja, sem a intervenção do usuário;
- **Pró-atividade**: um agente não apenas reage às mudanças no ambiente, mas também tem a iniciativa de prever determinadas circunstâncias e se prevenir delas;
- **Reatividade**: é a capacidade que o agente possui de reagir adequadamente às influências ou informações do seu ambiente;
- **Comunicação e cooperação**: em algumas oportunidades é conveniente ao agente interagir com o ambiente para realizar suas tarefas. Para realizar estas interações, é necessário que o agente possua capacidades de comunicação e cooperação. A comunicação permite que os agentes se comuniquem e troquem informações entre si. Porém, isto não é suficiente para resolver problemas complexos. Neste caso, torna-se necessário a cooperação entre vários agentes. A cooperação permite uma melhor solução para problemas que não podem ser resolvidos por um único agente;
- **Mobilidade**: é a capacidade que um agente pode possuir de migrar de um *host* para outro;
- **Caráter**: é desejável que os agentes possuam um bom comportamento externo, baseados na honestidade, confiança e segurança;
- **Raciocínio**: para que possa aprender e realizar com eficiência suas tarefas, um agente deve ter um certo grau de inteligência. De maneira genérica, a inteligência de um agente pode ser formada pelos seguintes componentes: base de conhecimento, capacidade de raciocínio sobre a base e a habilidade de aprender e se adaptar ao ambiente.

2.6.3 Tipos de agentes

De acordo com (NWANA; HEATH, 1996), há alguns critérios para se classificar um agente. Os agentes podem ser estáticos ou móveis, dependendo da sua capacidade de deslocamento na rede. Outra classificação possível de um agente é quanto ao modelo de representação do ambiente, podendo ser reativo ou deliberativo.

Agentes deliberativos

Os agentes deliberativos possuem um modelo simbólico do ambiente em sua base de conhecimento e têm a capacidade de raciocínio lógico como base das ações inteligentes. Para (BRENNER; WITTIG; ZARNEKOW, 1998), a representação do conhecimento neste tipo de agente é previamente modelada e constitui a parte mais importante da base de conhecimento. Além disso, usa sua base para alterar o seu estado interno, que é composto por três componentes básicos: crença, desejo e intenção.

Agentes reativos

Segundo (BRENNER; WITTIG; ZARNEKOW, 1998), os agente reativos não possuem uma representação simbólica interna do seu ambiente e a capacidade de raciocínio complexo é limitada. Assim, consegue-se criar agentes compactos, simples e flexíveis. Para este tipo de agente, a inteligência é fruto da interação dele com outros agentes e com o ambiente, e não de modelos internos complexos. O agente reativo observa o ambiente, reconhece as mudanças que ocorrem e atua sobre ele. Este esquema foi denominado de modelo estímulo-resposta (WOOLDRIDGE; JENNINGS, 1995). Seguindo este modelo, é possível monitorar continuamente um ambiente e atuar sobre ele quando for interessante para o sistema. Este tipo de agente possui características que se encaixam neste trabalho.

2.6.4 Arquitetura do agente

(DAVIDSSON, 1992) apresenta uma arquitetura (figura 2.6) que descreve todos os agentes autonômicos.

Nesta figura, os sensores são responsáveis por receber informações do ambiente e providenciar dados para o mecanismo de inferência; este, por sua vez, é o cérebro do agente inteligente, pois no momento em que recebe algum estímulo (por exemplo, ocorrência de um evento), opera sobre as regras pré-definidas e determina como melhor reagir ao estímulo; a base de conhecimento é onde o agente armazena seu conhecimento; os atuadores são os responsáveis pela execução das ações definidas pelo mecanismo de inferência.

Analisando a arquitetura descrita nesta seção, vê-se que existe uma grande similaridade com o gerente autonômico, descrito na seção 2.5. Devido a esta semelhança, fica claro que o gerente autonômico pode ser implementado por agentes de software.

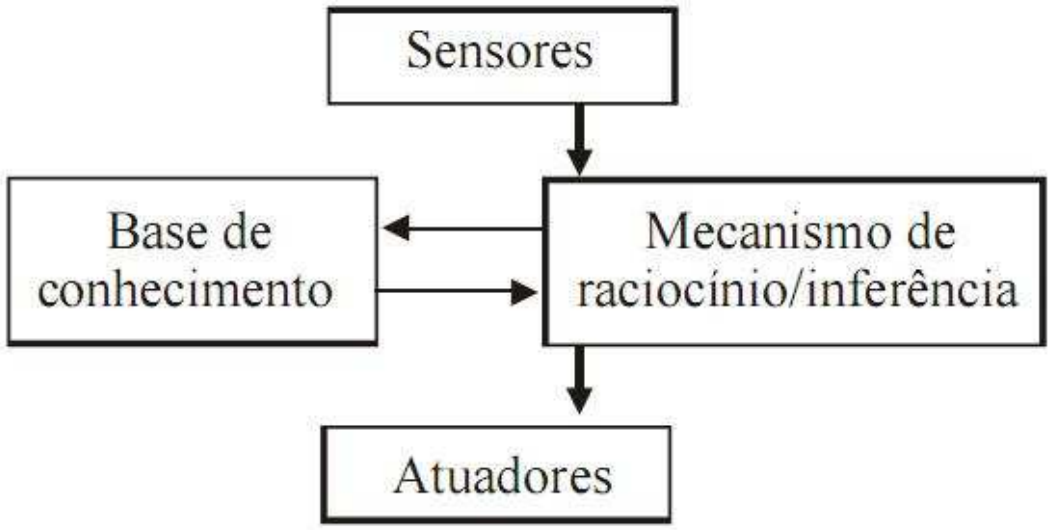


Figura 2.6: Arquitetura de um agente (DAVIDSSON, 1992)

2.6.5 Sistemas multi-agentes

Com o tempo foi ficando claro que a visão de agentes como se fossem entidades isoladas é muito simplista, pois a quantidade de conhecimento que um agente consegue manipular é limitada. Para (SYCARA, 1998), os sistemas multi-agentes proporcionam uma estrutura modular necessária para solucionar problemas complexos. Isto é feito dividindo o problema em partes menores, tratáveis pelos agentes, que assim são especialistas em uma parte específica do problema.

Quando se fala em múltiplos agentes, não importa apenas os aspectos relativos à inteligência de cada um, mas também aspectos sociais como a colaboração entre os vários agentes.

Segundo (ASSUNCAO, 2004), algumas características dos sistemas multi-agentes são:

- Cada agente tem informação ou capacidades incompletas para solucionar o problema geral e possuem uma visão limitada sobre ele;
- Não existe um controle global do sistema;
- Os dados e informações são descentralizados;
- A computação é assíncrona.

Neste momento atual, no qual os sistemas autonômicos são chamados para resolver problemas em ambientes complexos, percebe-se que a interação de vários agentes pode prover uma implementação possível para este cenário.

2.7 Considerações sobre o capítulo

Neste capítulo foram discutidos alguns fundamentos e características da computação autonômica, que é de suma importância para este trabalho. Como exposto no decorrer do capítulo, a computação autonômica é uma solução possível para responder à pergunta levantada na introdução: *como fazer com que um ambiente totalmente heterogêneo de grande complexidade como as grandes computacionais, não fiquem sujeitos à gerência manual, que muitas vezes é ineficiente?*

O capítulo começa trazendo definições e características da computação autonômica. Explica a origem do termo, derivado da tentativa de fazer com que um sistema de computação tenha funções semelhantes às desempenhadas pelo sistema nervoso humano.

A principal característica dos sistemas de computação autonômicos é a capacidade de auto-gerenciamento, liberando os administradores dos detalhes de operação e manutenção. Estas características estão resumidas na tabela 2.3. Possuindo as características citadas na tabela, um sistema autonômico poderá manter e ajustar suas operações de acordo com a mudança dos componentes e fluxos de trabalhos.

Tabela 2.3: As quatro principais características do auto-gerenciamento

Conceito	**Estágio atual**	**Computação Autonômica**
Auto-configuração	Os sistemas das empresas de TI são heterogêneos. A instalação, configuração e integração destes sistemas consomem muito tempo e estão sujeitas a erros	A auto-configuração dos componentes é feita seguindo políticas de alto-nível
Auto-otimização	Os sistemas apresentam muitos parâmetros que devem ser ajustados manualmente. Do ajuste destes parâmetros depende toda a otimização do sistema	Os componentes procuram continuamente melhorar sua performance e eficiência
Auto-regeneração	A determinação de um problema em sistemas complexos pode levar vários dias de trabalho dos programadores	Os sistemas localizam, diagnosticam e reparam automaticamente problemas tanto de hardware quanto de software
Auto-proteção	A detecção e a recuperação de ataques e de falhas em cascata de software ou hardware são realizadas de modo manual	Os sistemas defendem-se automaticamente de ataques e falhas em cascata. Pode-se usar experiências anteriores para antecipar e prevenir falhas generalizadas

Depois de explicar as principais características dos sistemas autonômicos, o capítulo detalha (seção 2.4) as arquiteturas da computação autonômica. A primeira deles é o elemento

autonômico, unidade funcional dos sistemas autonômicos. Um elemento autonômico consiste de um ou mais elementos gerenciados, acoplados a um único gerente autonômico que controla os elementos gerenciados. Outra arquitetura importante é o Blueprint IBM, introduzida pela empresa como uma arquitetura genérica para a construção dos ACS (figura 2.4). A arquitetura sugerida pela IBM não foi criada para trabalhar com ambientes de grade. Porém, essa arquitetura é a precursora na área e apresenta características fundamentais para fornecer autonomia a qualquer sistema computacional.

Para um componente ser auto-gerenciável, é necessário que o mesmo possua métodos de coleta autonômicos, meios para analisar estas informações para determinar se algo deva ser mudado. Além disso, deve possuir modos de criar planos, ou seqüências de ações, que especificam as mudanças necessárias, e deve estar apto para executar estes planos. Quando estas funções podem ser automatizadas, um inteligente laço de controle é formado. Como é o gerente autonômico o componente que implementa o laço de controle, por dedução, conclui-se que é ele o responsável pela autonomia do sistema.

O gerente autonômico é composto de quatro partes: monitoramento, análise, planejamento e execução, como mostra a figura 2.5. Os detalhes das quatro partes estão resumidas na tabela 2.4.

Tabela 2.4: Resumo das funções das áreas que formam o gerente autonômico

Áreas	**Funções**
Monitoramento	Fornece mecanismos que coletam, agregam, filtram e geram detalhados relatórios dos dados coletados (como as métricas e topologias) para o Gerente Autonômico.
Análise	Provê meios para correlacionar e modelar as complexas situações que o sistema pode ter. Estes mecanismos permitem ao Gerente Autonômico aprender sobre o ambiente e ajudam a predizer futuras situações.
Planejamento	Fornece os mecanismos que constroem as ações necessárias para alcançar os objetivos. O mecanismo planejador usa informações baseadas em políticas para guiar seu trabalho.
Execução	Provê modos que permitem ao sistema controlar a execução das ações decididas no Planejamento para atualizações dinâmicas.

A seção 2.5.6 discute os dois tipos de arquiteturas de software existentes para computação autonômica: fortemente e fracamente acoplada. A primeira é construída usando agentes inteligentes, formando um sistema multi-agentes, no qual cada agente tem seus próprios obje-

tivos, guiando suas próprias ações. Na segunda, os componentes individuais não são por si só autonômicos. Existe uma infra-estrutura autonômica que é separada do ambiente de execução e trata as informações de todos os componentes do sistema. Deve-se ressaltar que a abordagem fortemente acoplada apresenta uma característica inerentemente distribuída, assim como os ambientes de grades computacionais.

Finalizando o capítulo, são apresentadas informações sobre agentes e sistema multi-agentes. Este tipo de sistema é a forma mais adequada de se alcançar um sistema autonômico, já que apresenta abordagem fortemente acoplada. Um agente é uma entidade que percebe o mundo através de seus sensores e atua sobre ele por meio dos atuadores. Porém, a quantidade de conhecimento que um agente consegue manipular é limitada. Por isso, os sistemas multi-agentes tornam-se tão importantes, pois não importa os aspectos relativos à inteligência de cada um, mas sim, os aspectos sociais que farão com que o sistema aja de acordo com o interesse global.

Com o intuito de dar prosseguimento ao estudo teórico, no próximo capítulo, serão discutidas as grades computacionais, nas quais o sistema proposto neste trabalho será ambientado e testado.

3 *Grades Computacionais*

A crescente necessidade de aumento da capacidade computacional, principalmente para resolver problemas não convencionais, sempre esteve presente na informática. A partir de 1980, começa a ser utilizada uma técnica que envolve agrupamento de vários computadores, resultando na emulação de um computador multiprocessado (FOSTER, 2000). Estes são chamados de *clusters* e utilizados como uma alternativa eficiente e muito mais acessível do que os supercomputadores, sendo utilizados em muitos centros de supercomputação até hoje.

No final dos anos 90, uma outra abordagem foi proposta (FOSTER, 2000). O nome era grade computacional (*grid computing*) e sua principal característica foi descrever uma arquitetura que possibilitava a um conjunto geograficamente distribuído a capacidade de suportar aplicações de larga escala.

3.1 Definição

O termo grade advém da analogia feita com o sistema de distribuição de energia elétrica (*Eletrical Power Grid*), o qual é capaz de proporcionar aos seus usuários um acesso transparente, fácil e barato à eletricidade através de uma malha de fios e pontos de transmissão, escondendo do usuário detalhes de como a energia é gerada. A corrente elétrica possivelmente atravessou o estado, ou o país inteiro, proveniente de uma usina que utiliza água, vento, ou energia nuclear. O consumidor, entretanto, não tem interesse em saber como a eletricidade foi gerada, deseja apenas utilizá-la.

Outra analogia possível de ser feita é com a Internet. Nela toda a complexidade por trás da web fica oculta para os usuários. Quando se recorre a um *website* qualquer, não é necessário que se conheça a localização do servidor, ou o procedimento completo que seu navegador executou para encontrá-lo. Esta é exatamente a proposta da computação em grade, ou seja, oferecer recursos computacionais aos usuários da grade, sem que seja necessário conhecer detalhes como a localização do recurso ou o protocolo utilizado.

A grade fornece uma visão transparente do sistema como se o conjunto de recursos (*hardware* e *software*) que a formam fosse um único e poderoso computador (IBM, 2001). A idéia básica da computação em grade consiste no compartilhamento coordenado de recursos, e resolução de problemas em organizações virtuais multi-institucionais e dinâmicas (FOSTER; TUECKE, 2001).

Para (DANTAS, 2005), as grades representam uma forma estendida dos serviços web, provendo não somente a comunicação automática entre processos, mas também permitindo que recursos computacionais possam ser compartilhados.

Além disso, a computação em grade possibilita a integração de redes de diferentes instituições de forma dinâmica, para compartilhamento de ciclos de processador, espaço em disco, acesso a banco de dados, software, arquivos, ou qualquer outro tipo de recurso, como microscópios eletrônicos ou telescópios. A grande motivação dos pesquisadores é tornar disponível um excelente poder computacional para os participantes da grade.

(FOSTER, 2002) descreve três pontos importantes e fundamentais para as grades, definindo-as como um sistema que:

- coordena recursos que não estão sujeitos a um controle centralizado, ou seja, recursos em diferentes unidades administrativas são coordenados e integrados por meio de uma grade;
- usa protocolos e interfaces de padrão aberto e para propósito geral, ou seja, uma grade é construída e gerida por protocolos e interfaces de propósito geral que atendem a questões fundamentais como autenticação, autorização, descobrimento e acesso aos recursos;
- entrega qualidade de serviço não trivial, ou seja, um grade permite por meio de seus recursos combinados entregar diferentes qualidades de serviço, como tempo de resposta, disponibilidade, banda passante, segurança entre outros. Baseado nesta evolução das definições de *grid computing*, presentemente, a grade possui definições bem próximas às anteriores, sendo uma delas: "*Grid Computing* habilita organizações a trocar e compartilhar informações e recursos computacionais entre departamentos e organizações de forma segura e eficiente"(GRIDFORUM, 2006).

3.2 Origem e futuro

Segundo (ROURE; JENNIGS; SHADBOLT, 2003), as primeiras iniciativas de computação em grade visavam interligar supercomputadores e centros computacionais dispersos geograficamente. Como precursores deste cenário podem ser citados dois projetos: Fafner (FAFNER,

) e Globus (FOSTER; KESSELMAN, 1997), atualmente um dos projetos mais expressivos em *middleware* para computação em grade muito usado por americanos e europeus.

Após esta abordagem inicial, as grades passaram a ser usadas como uma plataforma de execução para a resolução de aplicações de grande desafio que até este momento usava *clusters* e outras arquiteturas paralelas para este fim. As grades permitem reunir uma grande quantidade de recursos a um custo muito menor que os oferecidos pelos *clusters*. Por isto, vários outros projetos apareceram propondo *middlewares* que ofereciam computação em larga escala.

Atualmente, é possível encontrar grades em vários escopos diferentes, executando diferentes aplicações. Para (FOSTER; TUECKE, 2001), futuramente, as grades computacionais serão capazes de proporcionar o acesso a recursos computacionais de modo análogo ao sistema elétrico, sendo necessário apenas que o usuário se conecte na grade e obtenha os recursos desejados.

3.3 Tipos de grades

Existem várias aplicações práticas nas quais as grades podem oferecer grandes benefícios. De acordo com (SKILLICORN, 2002), existem quatro tipos de grades: *grids* computacionais, grades de acesso, grades de dados e grades datacêntricos. Cada um dos tipos oferece algumas vantagens, como será descrito a seguir:

- grades computacionais: através de uma única interface, usuários utilizam o sistema como um grande computador paralelo, executando as requisições dos usuários com grande aumento de desempenho;
- grades de acesso: oferecem um ambiente no qual usuários de diferentes organizações interagem como se estivessem na mesma plataforma de hardware. Este tipo não possui o desempenho como objetivo primário;
- grades de dados: permitem que grandes conjuntos de dados sejam armazenados em repositórios e manipulados com a mesma facilidade com que se manipula pequenos arquivos atualmente. O objetivo principal deste tipo de grade é a disponibilidade de dados;
- grades datacêntricas: este tipo difere da grade de dados, no sentido que ao invés de mover os dados para a computação move a computação para os dados, em casos nos quais os dados devem permanecer imóveis. Uma aplicação das grades datacêntricos é a mineração de dados distribuídos (*distributed data mining*).

3.4 Diferenças entre grades e outras abordagens semelhantes

O ambiente de grades computacionais apresenta semelhanças com outras abordagens mais antigas como os agregados computacionais (*clusters*) e *peer-to-peer* (P2P). Apesar das características em comum, estas abordagens possuem diferenças significativas. Esta seção visa esclarecê-las.

3.4.1 Grades x agregados

Alguns fatores diferenciam os agregados computacionais das grades. Geralmente, um agregado está restrito a um único domínio administrativo e seus recursos são muito mais homogêneos que os apresentados pela grade. Além disso, é comum concentrar os recursos dos agregados em uma mesma área geográfica, ao contrário das grades, nas quais os recursos estão distribuídos geograficamente. A forma como os recursos são regulados é outro diferencial. Nas grades, não há nenhuma autoridade reguladora no uso de recursos disponíveis (como nos agregados) ou serviços providos por organizações virtuais (como em uma configuração de grade) (DANTAS, 2005).

Outro fato importante observado é que os recursos de um agregado estão dedicados exclusivamente para seu uso próprio, diferentemente do que ocorre com as grades, nas quais os recursos podem estar sendo utilizados em um momento pela grade, e em outro, pelo proprietário real do recurso.

A escalabilidade das abordagens também é bem diferente. Enquanto os *middlewares* empregados nos agregados permitem a utilização de apenas dezenas ou centenas de unidades de processamento, as grades podem possuir milhares, permitindo ser utilizadas como plataforma de processamento de aplicações de grande desafio.

Mais informações sobre as diferenças entre agregados computacionais e grades podem ser obtidas em (DANTAS, 2005).

3.4.2 Grades x P2P

De acordo com (FOSTER; IAMNITCHI, 2003), as grades e os agregados compartilham algumas características em comum. As principais são:

- tanto a grade como o P2P visa a organização do compartilhamento de recursos dentro das comunidades virtuais;

- ambas organizam o compartilhamento através da criação de estruturas adicionais que usam as estruturas existentes.

Apesar das semelhanças, as duas tecnologias divergem em vários pontos, dos quais pode-se destacar:

- as grades estão mais preocupadas com a infra-estrutura e não com as falhas, enquanto que P2P dá mais importância às falhas do que com a estrutura;
- a grade fornece recursos sofisticados, enquanto o P2P tem limitações quanto ao tipo de recurso compartilhado;
- as aplicações das grades variam em escopo e tipo, enquanto as aplicações P2P resolvem problemas de compartilhamento bastante específicos;
- as grades tentam fornecer aspectos de qualidade de serviço. Já o P2P, simplesmente, ignora este aspecto.
- as grades conectam recursos mais sofisticados e poderosos que os recursos interligados por redes P2P. Exemplo disso, é a presença de agregados nas redes interligadas por grades;
- a grade se preocupa com a criação de uma infra-estrutura de múltiplo propósito, enquanto o P2P focaliza a integração de recursos mais simples.

Mais detalhes sobre P2P podem ser obtidos em (COULOURIS; DOLLIMORE; KINDBERG, 2005).

3.5 Grades orientados a serviços

As grades que apresentam a abordagem orientada a serviços possuem vantagem se comparadas com as tradicionais. A principal delas é a capacidade de proporcionar serviços com fins comerciais e científicos para qualquer usuário, independente do momento ou do lugar de acesso. Segundo (FOSTER; TUECKE, 2001), a estruturação das grades neste cenário de orientação por serviços fornece funcionalidades importantes para que seja possível a formação das organizações virtuais dinâmicas.

Um serviço possui sempre um proprietário, ou seja, é oferecido por um indivíduo ou instituição, e está hospedado em algum lugar. O proprietário oferece um serviço para ser consumido

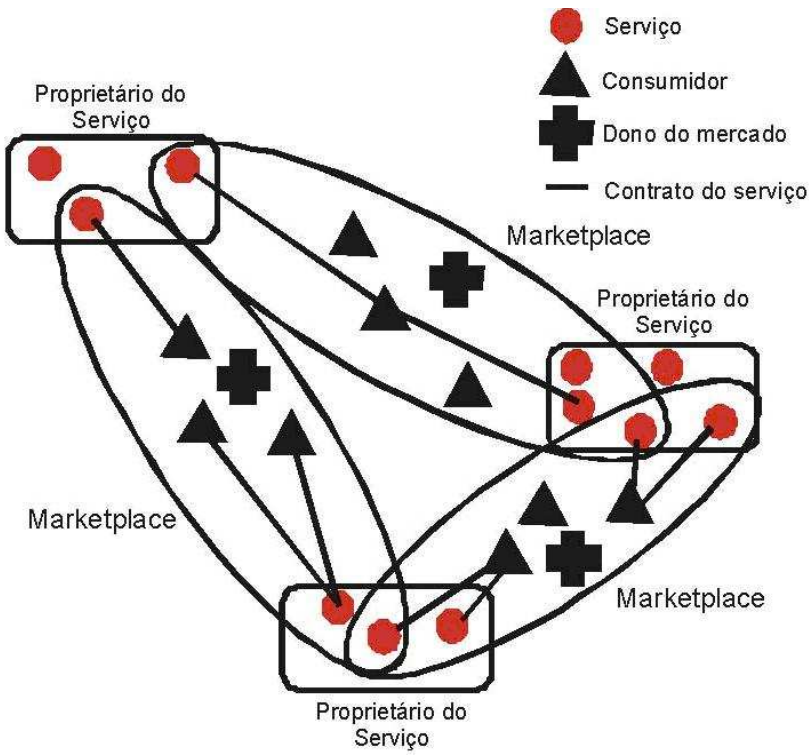


Figura 3.1: Infra-estrutura orientada a serviços (ROURE; JENNIGS; SHADBOLT, 2003)

por alguém. As políticas para disponibilizar os serviços podem ser diferentes para cada proprietário. Alguns podem disponibilizar seus serviços livremente na Internet, e podem ser utilizados sem nenhum custo, enquanto outros podem requerer algum valor ou pagamento para que seus serviços sejam utilizados. Os consumidores dos serviços são entidades que invocam a execução daqueles e os critérios para utilização ou se eles alocam estes serviços para oferecê-los a outros consumidores fica a critério da estratégia adotada pelos consumidores (ASSUNCAO, 2004).

Segundo (ROURE; JENNIGS; SHADBOLT, 2003), os componentes chaves de uma arquitetura orientada por serviços são apresentados na figura 3.1: proprietários dos serviços (envolvidos pelos retângulos) oferecem serviços (círculos com preenchimento) para consumidores (triângulos com preenchimento) sob certos contratos (linhas sólidas entre produtores e consumidores). Cada interação com o dono do recurso ocorre em um ambiente baseado em mercado (denotado pelas formas ovais) nas quais as regras são estabelecidas pelo dono do comércio (forma em cruz). O dono do mercado pode ser uma das entidades no ambiente de mercado (um produtor ou consumidor) ou pode ser uma terceira parte neutra.

Esta visão de negociação de serviços da figura 3.1 adota uma visão de comércio (*marketplace*). Nem sempre esta visão precisa ser seguida. Existem casos nos quais a entrada de entidades pode ser controlada e restrita a determinados grupos ou indivíduos.

3.6 Virtualização de Recursos

A virtualização de recursos (JOSEPH; ERNEST, 2004) é o encapsulamento de vários tipos de recursos que fornecem uma funcionalidade através de uma interface comum a todas as implementações. Usando esse conjunto de abstrações, o usuário pode agregar diversos tipos de computadores na sua grade. Para efetuar a agregação, o usuário deve informar ao *middleware*, quais são os recursos que compõem sua grade e como acessá-los (transferência de arquivos e execução remota). Semelhante à forma como são descritos os recursos que compõem a grade, o usuário descreve as tarefas que formam a aplicação.

3.7 *Middleware* para a computação em grade

O *middleware* é a camada de software colocada entre o sistema operacional e as aplicações, proporcionando uma série de serviços requeridos pelas aplicações. Para (DANTAS, 2005), esta camada deve fornecer protocolos que permitam a múltiplos elementos (servidores, ambientes de armazenamento, redes e outros elementos) participar de um ambiente de grade unificado.

O *middleware* tem sido largamente usado em ambientes de computação distribuída e, em um ambiente de grade, ele pode ser usado para ocultar da aplicação, a heterogeneidade de recursos existentes no ambiente, proporcionando um acesso uniforme e seguro aos recursos. Ele compõe a camada que está em um nível mais baixo que as aplicações dos usuários e acima dos serviços e meios de transporte proporcionados pela infra-estrutura de rede.

De acordo com (FOSTER, 2000), as principais funções de um *middleware* são:

- mascarar o ambiente distribuído, oferecendo ao usuário ou ao desenvolvedor um sistema único;
- ocultar a heterogeneidade de dispositivos de hardware, de sistemas operacionais e de protocolos de comunicação;
- prover ao desenvolvedor interfaces padronizadas para que as aplicações desenvolvidas possam ser portáveis, reusáveis, interoperáveis e sejam facilmente construídas.

3.7.1 Principais *middlewares*

Existem alguns projetos com grande destaque na área de desenvolvimento de *middlewares* para computação em grades. Os principais são:

- O Globus (FOSTER; KESSELMAN, 1997) é um software de código aberto, desenvolvido pela Globus Alliance, que oferece um kit de ferramentas para desenvolver aplicações e sistemas de computação em grade. Apresenta suporte a colaboração que por meio dos protocolos da camada de recursos (GRAM, GRIP, GridFTP) pode obter e receber informações, além de controlar as tarefas, promovendo assim colaboração pela distribuição aos recursos. Também existe o suporte a alocação de recursos, provido pelo gerenciador de recursos (GRAM - *Globus Resource Allocation Manager*) que fornece uma interface para envio e monitoramento de tarefas;

- O Gridbus (GRIDBUS, 2005) do laboratório de pesquisa e desenvolvimento de software para computação em grade e sistemas distribuídos (GRIDS) da universidade de Melbourne na Austrália, é um pacote de código aberto utilizado para arquiteturas na implementação de grades de computadores para *eScience* e *eBusiness applications*. Para isso, utiliza outros *middlewares* como Globus, Unicore, Alchemi entre outros. Suporta colaboração, alocação de recursos e suporte a ambientes dinâmicos, providos por *middlewares* de baixo nível ou *core middlewares*, como os acima descritos e também usados no desenvolvimento de aplicações;

- O Legion (LEGION, 2006) é um *middleware* desenvolvido por um projeto da universidade da Virginia, definido como um meta-sistema baseado em objetos (recursos) com bilhões de hosts e trilhões de objetos vinculados por *links* de alta velocidade conectando redes, estações de trabalho, supercomputadores em um sistema que pode agregar diferentes arquiteturas, sistemas operacionais e localizações físicas. Suporta colaboração e alocação de recursos, que é realizada pelo LOA (Legion Object Addresses) que incorpora um endereço físico como o IP e pode distribuir estes recursos como multicast ou comunicação entre grupos de objetos;

- O UNIform Interface to COmputer REsources (UNICORE) (UNICORE, 2006) é um *middleware* que integra os recursos da computação em grade por meio de uma interface gráfica desenvolvida na linguagem JAVA. Oferece um ambiente pronto para execução de sistemas de grade, incluindo *software* e *hardware*;

- O Alchemi (LUTHER et al., 2005) é um framework que agrega computadores formando um ambiente de supercomputador virtual. Além disso, provê suporte para o desenvolvimento de aplicações;

- O OurGrid (BRASILEIRO et al., 2007) é *middleware* computacional baseado em P2P, aberto e em produção desde dezembro de 2004. Oferece poder computacional a qualquer

usuário interessado em se juntar à grade e executar suas aplicações paralelas. Seu poder computacional é obtido através dos recursos ociosos dos seus participantes e é compartilhado de tal forma que recebe mais recursos quem oferece mais recursos. Atualmente, a plataforma pode ser usada para executar aplicações do tipo *Bag-of-Tasks*, ou seja, aplicações paralelas cujas tarefas são independentes. Isto significa que as tarefas (partes da aplicação) executam paralelamente no grid, porém não se comunicam entre si. Por isso, este middleware está sendo usado com grande sucesso em aplicações científicas. Aplicações que fazem simulação, mineração de dados e renderização de imagem são exemplos de aplicações desse tipo;

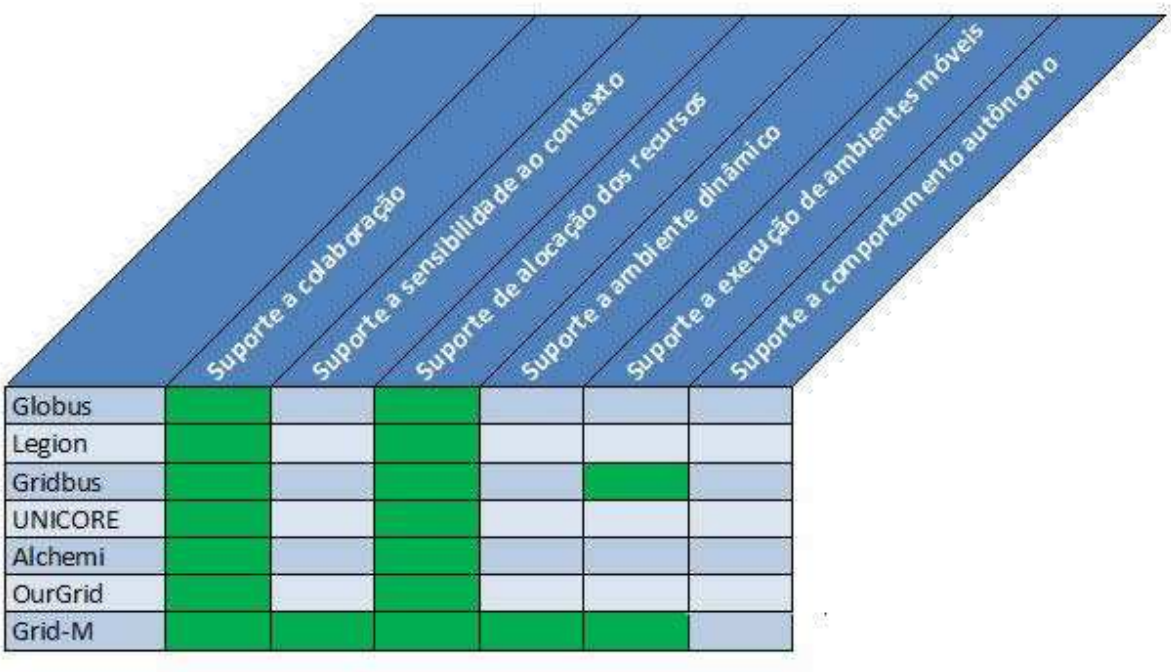
- O Grid-M (FRANKE et al., 2007), discutido com detalhes na seção 3.8, é um *middleware* desenvolvido pelo Laboratório de Redes e Gerência (LRG) da Universidade Federal de Santa Catarina, que visa a criação de um ambiente homogêneo, no qual todos os dispositivos são vistos como nós. Para criar esse ambiente, o *middleware* utiliza a virtualização de recursos (seção 3.6), trabalhando como uma biblioteca integrada em uma aplicação Java e provendo uma *Application Programming Interface* (API) para os nós. A biblioteca provê as funcionalidades básicas para criar nós em grades: comunicação (envia, espera e recebe tarefas), interface de serviço, interface de sensor, segurança, vários *hook-up* para integrar lógica externa e outros.

Todos os projetos citados anteriormente possuem semelhanças e diferenças. A tabela 3.1 mostra as características suportadas por eles. Observa-se que todos os *middlewares* listados suportam colaboração e alocação de recursos. Porém, apenas dois suportam execução de ambientes móveis, apenas um suporta sensibilidade ao contexto e nenhum oferece comportamento autonômico. Devido à complexidade das grades, torna-se necessário um *middleware* com suporte à autonomia.

3.8 Grid-M

Ficou claro na seção anterior que com os *middlewares* convencionais não é possível lidar com as características presentes em ambientes móveis e embarcados. Por isso, o Grid-M (LRG, 2006) é de suma importância para este trabalho. Para explicá-lo, esta seção é dedicada a detalhar sua arquitetura e principais características.

Tabela 3.1: Sumário dos principais *middlewares* e características suportadas



3.8.1 Definição

Este *middleware* visa a criação de um ambiente homogêneo, no qual todos os dispositivos são vistos como nós. Para criar esse ambiente, o Grid-M utiliza a virtualização de recursos (explicada com detalhes na seção 3.6). Com a intenção de garantir a portabilidade necessária, este projeto foi implementado na linguagem Java.

O Grid-M trabalha como uma biblioteca integrada em uma aplicação Java e provê uma *Application Programming Interface* (API) para os nós (FRANKE et al., 2006). A biblioteca provê as funcionalidades básicas para criar e integrar nós em grades, possibilitando dinamicidade e mobilidade, características fundamentais para que a grade consiga assimilar mudanças de topologia ocasionada pela movimentação, conexão e desconexão de um dispositivo.

3.8.2 Componentes da arquitetura

A figura 3.2 mostra os componentes da arquitetura do Grid-M (ROLIM, 2007). O sistema é composto por:

- Interface sensor: responsável pela troca de informações entre a arquitetura e o sensor. Esta interface garante a flexibilidade na implementação da comunicação com diferentes tipos de sensores;

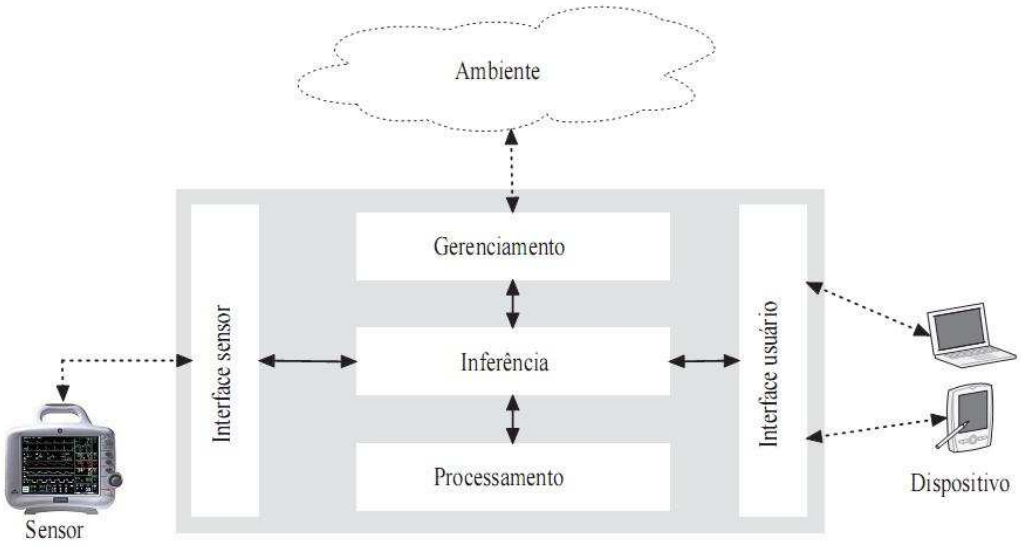


Figura 3.2: Componentes da arquitetura do Grid-M (ROLIM, 2007)

- Interface usuário: fornece mecanismos que permitem ao usuário interagir com a arquitetura. Este componente permite que um dispositivo envie tarefas para o sistema e receba os resultados da execução;
- Gerenciamento: responsável pela coordenação e descoberta de recursos, além da comunicação entre os dispositivos. Também assegura qualidade de serviço no processamento de tarefas e gerenciamento do dinamismo existente no ambiente;
- Inferência: responsável por inferir e converter os dados para o padrão utilizado pela arquitetura (as mensagens são formatadas em XML). Assim, este componente codifica e decodifica os dados usando padrões abertos e reconhecidos por todas as partes da arquitetura;
- Processamento: as tarefas dentro da arquitetura são definidas como estruturas de dados que caso sejam executadas geram um resultado. Assim, este componente é responsável pela execução das tarefas.

3.8.3 Camadas da arquitetura

As camadas delimitam tarefas, auxiliando na separação das responsabilidades. Além disso, fornecem ao sistema baixo acoplamento e alta coesão dos componentes.

A figura 3.3 mostra as camadas da arquitetura do Grid-M (ROLIM, 2007). O sistema é composto pelas seguintes camadas:

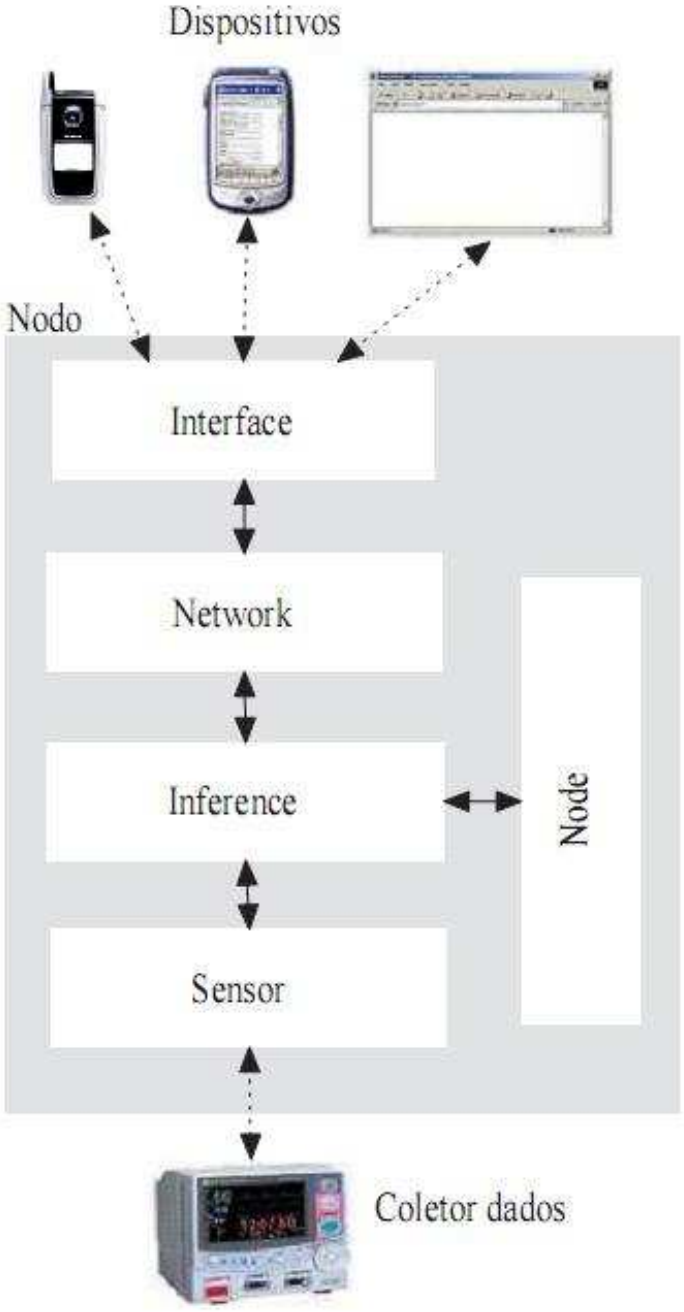


Figura 3.3: Camadas da arquitetura do Grid-M (ROLIM, 2007)

- Sensor: é a camada mais baixa, responsável por interagir com os dispositivos embarcados existentes no ambiente. Sua principal função é dar flexibilidade à incorporação de novos dispositivos. Esta camada possui métodos com a função de coletar os dados capturados pelo dispositivos;

- Inference: camada responsável pela inferência e conversão dos dados no formato padrão da arquitetura;

- Node: esta camada possui métodos para o gerenciamento interno de cada nó. Além disso, fornece métodos relacionados com a criação, escalonamento, coordenação e execução de tarefas;

- Network: camada responsável pela comunicação entre os nós (roteamento e manipulação do protocolo de comunicação). A comunicação ocorre através de um protocolo pervasivo, garantindo transparência e organização dos dados. O protocolo escolhido foi o HTTP, pois sua implementação é possível em diversos tipos de dispositivos, além de ser considerado um padrão de fato;

- Interface: é a camada acessada pelo usuário. Proporciona a integração do usuário com a arquitetura, recebendo tarefas para serem executadas pelas camadas inferiores, além de formatar os resultados das tarefas que serão apresentadas de acordo com o tipo do dispositivo.

Na arquitetura, as funções exercidas por uma camada servem como base tanto para a camada superior quanto para a inferior. Isto garante flexibilidade e simplicidade à arquitetura.

3.9 Redes de sensores

Sensores são compreendidos como um sistema micro-eletromecânico (MEMS) que possuem uma unidade central de processamento *Central Processing Unit* (CPU), memória e um transmissor sem fio (*wireless transceiver*). Tais sensores podem monitorar um ambiente e até mesmo afetar, interagir com este ambiente dando margem a aplicações nas áreas da saúde, tráfego, segurança, bélico entre outros.

De acordo com (MALLADI; AGRAWAL, 2002), uma rede de sensores sem fio é formada por um conjunto de sensores capazes de detectar e transmitir através de ondas de rádio as características físicas do fenômeno ou ambiente no qual estão imersos. Depois de coletados e transmitidos, os dados são armazenados em uma base de dados para posterior consulta.

Devido às características particulares deste ambiente, como necessidade de economia de energia, constantes alterações de topologia, eleições de nós líderes, roteamento eficiente e tolerância a faltas, um comportamento auto-configurável deve estar presente. Para (ROLIM, 2007), estas características fazem com que sejam necessários protocolos e estudos específicos para redes de sensores que não possam ser aplicados em outros sistemas distribuídos.

Aplicações de redes de sensores possuem grande disseminação tanto no âmbito acadêmico quanto comercial. Isto se deve em grande parte pela capacidade de vincular o mundo real com o mundo virtual (computacional). Na área de desenvolvimento bélico, já são vistos sistemas em que forças aéreas, terrestres e marítimas se intercomunicam e têm visualização de posicionamento real das unidades através de uma rede de sensores, no qual cada unidade possui um sensor que recebe e envia informações.

Como explica (FRANKE et al., 2006), uma das grandes funcionalidades do Grid-M é dar suporte aos ambientes de redes de sensores.

3.10 Considerações sobre o capítulo

No decorrer do capítulo, viu-se que a computação em grade é o resultado da evolução de uma série de tecnologias. Em um período de mais de uma década, vários projetos foram apresentados na área de computação em grades, melhorando significamente a computação de alto desempenho.

Não se deve esquecer que as grades surgiram para tratar aplicações de grande desafio e que, atualmente, há vários projetos com os mais variados fins. Isto mostra a grande evolução que existiu nesta área.

Junto com esta nova tendência, percebe-se uma inclinação para um ambiente orientado a serviços, proporcionando independência de momento e de lugar de acesso.

Foi exposto que um *middleware* capaz de suportar ambiente computacional com a presença de componentes pervasivos e outros sistemas embarcados deve oferecer computação distribuída em larga escala que integre sensores e dispositivos móveis.

A seção 3.7.1 mostra que o único *middleware* dentre os citados capaz de suportar componentes móveis com flexibilidade e dinamismo, levando em consideração a sensibilidade ao contexto, é o Grid-M. A seção 3.8 detalha os principais componentes e camadas do *middleware* Grid-M, projeto que é de grande importância para este trabalho.

Para finalizar, é explicado o que são as redes de sensores, quais são as suas principais

características e aplicações. Deve-se lembrar que uma das grandes funcionalidades do Grid-M é suportar a criação de uma rede de sensores.

Porém, nem mesmo este, possui comportamento autonômico. No próximo capítulo, será detalhada a proposta deste trabalho, mostrando a arquitetura proposta, os algoritmos de roteamento propostos e as modificações realizadas no Grid-M com a finalidade de torná-lo autogerenciável.

4 *Sistema para Gerência Autonômica de Grades Computacionais*

Como descrito no capítulo 1, a computação pervasiva é considerada o novo paradigma do século XXI, que visa fornecer uma computação onde se deseja e como se deseja, através da virtualização de informações, serviços e aplicações. Este ambiente computacional consiste em uma grande variedade de recursos de diversos tipos, móveis ou fixos, aplicações e serviços inter-conectados. Além disso, estes recursos são disponibilizados de maneira dinâmica e as aplicações necessitam ser construídas de forma distribuída e adaptativa ao contexto (BALEN; FRANKE; WESTPHALL, 2008).

Um *middleware* capaz de suportar este novo ambiente computacional deve oferecer computação distribuída em larga escala que possa integrar sensores e dispositivos móveis, levando sempre em consideração o dinamismo do ambiente e a sensibilidade do contexto.

O único dos citados com estas características é o Grid-M. Porém, assim como os outros, não oferece comportamento autonômico. As grades computacionais são reconhecidas como um ambiente computacional dinâmico e heterogêneo, e, apesar disso, a configuração e a gerência é realizada pelo esforço humano, ficando sujeita à lentidão para tomada de decisão ou até mesmo, erros por parte dos administradores. A fim de tratar este problema, necessita-se de uma solução na qual o trabalho de gerência não fica a cargo somente de administradores humanos.

Este trabalho propõe um sistema para este tipo de ambiente, oferecendo a oportunidade de criar uma grade computacional com gerência autonômica.

Antes de entrar em detalhes da proposta, será feito um estudo das iniciativas nas áreas de sistemas autonômicos, grades de computadores e dos trabalhos resultantes da união destas duas.

4.1 Trabalhos Relacionados

O sistema sugerido neste trabalho visa unir áreas que estão sendo alvo de muito pesquisa acadêmica como os sistemas autonômicos e as grades computacionais. Porém, a união destas duas iniciativas ainda é nova e existem raros trabalhos com este foco, destacando o caráter inovador desta proposta.

A computação autonômica é uma área de pesquisa multidisciplinar, por isto é possível encontrar vários projetos de outras áreas que não os das ciências da computação. Exemplo disto é o uso de computação autonômica na indústria automobilística, como ocorre nos sistemas ABS. No campo das ciências da computação, encontram-se várias iniciativas, tanto da área acadêmica quanto industrial, sendo algumas delas:

- *Ocean Store* (KUBIATOWICZ et al., 2000): proposta pela UC Berkeley, permite a bilhões de usuários um modo de armazenamento global de dados, no qual a otimização é feita de maneira autonômica;
- *AntHill* (BABAOGLU; MELING; MONTRESOR, 2002): projeto da Universidade de Bolonha que permite criar, implementar e medir aplicações P2P baseadas em sistemas adaptativos multi-agente;
- *Recovery-Oriented Computing* (PATTERSON et al., 2002): iniciativa da UC Berkeley/ Stanford que permite a modelagem de sistemas com alta confiabilidade para os serviços da Internet, fornecendo suporte à recuperação de falhas inesperadas;
- *Autonomia* (HARIRI, 2007): criada pela Universidade do Arizona, provê mecanismos para desenvolver aplicações autonômicas que permite especificar mecanismos de controle e gerência para manter requisitos de qualidade de serviço;
- *eBiquity* (GUPTA; JOSHI; FININ, 2008): projeto da Universidade de Baltimore que explora as interações entre sistemas móveis, pervasivos, multi-agentes e de inteligência artificial;
- *SMART* (LOHMAN; LIGHTSTONE, 2002): proposta pela IBM, reduz a complexidade e melhora a qualidade de serviço através de avançadas capacidades de auto-gerenciamento dentro de um ambiente de banco de dados;
- *Oceano* (IBM, 2005b): também proposto pela IBM, permite que se modele e desenvolva um protótipo de estruturas gerenciáveis e escaláveis para sistemas computacionais de larga escala;

- *AutoAdmin* (MICROSOFT, 2005): criado pela Microsoft, realiza auto-adaptação e auto-administração nos sistemas de banco de dados, permitindo que o sistema monitore o uso das suas aplicações e se adapte aos requisitos das mesmas;

- N1 (MICROSYSTEMS, 2008): desenvolvida pela SUN, gerencia os servidores de dados, propiciando virtualização de recursos, provisionamento de serviços e técnicas autonômicas baseadas no uso de políticas;

- (MALATRAS; HADJIANTONIS; PAVLOU, 2007) descrevem a modelagem e implementação de um sistema sensível ao contexto baseado em políticas que permitem auto-gerenciamento em redes móveis *ad hoc* (MANETs). A idéia chave é suportar auto-configuração, apresentando características adaptativas diante das variações das condições modeladas. A adaptação é feita seguindo políticas de alto-nível definidas por administradores que visam solucionar objetivos particulares;

- (YOU; CHOI; HAN, 2008) descrevem uma arquitetura autonômica para PDAs e *smart phones* baseada na auto-reconfiguração e auto-regeneração. A primeira provê reconfiguração dinâmica do *middleware* à medida que ocorrem mudanças nos componentes que compõem o ambiente, se preocupando também com a eficiência na execução das aplicações. A auto-regeneração provê mecanismos de recuperação do sistema e a reinstalação de aplicações e componentes do *middleware* contra erros que venham a ocorrer em tempo de execução.

Outro ponto fundamental de pesquisa neste trabalho são os *middlewares* para ambientes de grades computacionais, principalmente aqueles que suportam computação distribuída em larga escala e que integrem sensores e dispositivos móveis. Uma apresentação aprofundada sobre os principais *middlewares* foi feita na seção 3.7.1. Além destes trabalhos, há outros que merecem destaque por estarem ligados à concepção do Grid-M:

- (LIM et al., 2005) propõem uma arquitetura para grades de sensores, chamadas de *Scalable Proxy-based aRchItecture for seNsor Grid (SPRING)*. As grades de sensores combinam duas áreas promissoras discutidas no capítulo anterior, estendendo o paradigma de computação em grades para o compartilhamento de recursos em redes de sensores sem fio. Além de modelar a arquitetura, os autores implementam um protótipo para estudar os sensores e melhorar a modelagem da arquitetura;

- O *Grid Mobile Service*, projeto integrante do GridLab (ALLEN et al., 2002), permite que aplicações executadas em dispositivos móveis possam acessar os recursos de uma grade

fixa através de *gateways*. Porém, este projeto possui restrições: os clientes móveis só podem realizar consulta aos dados, não podendo compartilhar recursos;

- (GONZáLEZ-CASTA et al., 2002) descrevem a plataforma Condor que possui arquitetura baseada em camadas de forma que a reutilização seja facilitada. O seu principal objetivo é a incorporação de dispositivos móveis como clientes para a submissão de tarefas para as grades;

- (LOPES; SILVA; SOUSA, 2005) descrevem a modelagem, implementação e resultados da arquitetura MAG (*Mobile Agents for Grid Computing Environments*). A MAG executa aplicações carregando dinamicamente seus códigos nos agentes móveis. Esta arquitetura permite que os agentes sejam realocados dinamicamente entre os nós da grade através de um mecanismo de migração transparente chamado MAG/Brakes. Além disto, inclui mecanismos de tolerância a falhas de aplicações, de grades pervasivas e dados.

A junção dos temas grades computacionais sensíveis ao contexto com computação autonômica, ainda foi pouco explorada pela comunidade acadêmica, o que torna este trabalho mais desafiador. Todavia, existem trabalhos que vão ao encontro, se não em todos, mas da grande maioria das características apresentadas pela arquitetura proposta neste trabalho. Entre eles, encontram-se:

- (BECKSTEIN et al., 2006) apresentam a arquitetura SOGOS voltada para dar suporte à auto-organização em grades computacionais. Esta é capaz de trabalhar com ambientes dinâmicos através de informações semânticas (metadados) que descrevem as organizações envolvidas, papéis, direitos e capacidades dos agentes participantes e também da forma que eles interagem para solucionar o problema. Com base nos metadados é que as decisões de sistema são tomadas;

- (BRENNAND et al., 2007) apresentam o AutoMan, um sistema cujo objetivo é fornecer um certo nível de gerenciamento automático para grades computacionais entre pares. Além disso, busca-se otimizar o uso de recursos na grade, ao mesmo tempo que as atividades de gerência sejam simplificadas. Como base para a arquitetura autonômica, o AutoMan utiliza a grade OurGrid (BRASILEIRO et al., 2007), detalhada na seção 3.7.1;

- (LIU et al., 2005) propõem uma arquitetura autonômica para que se possa tratar a heterogeneidade e dinamismo do ambiente de grades. Esta arquitetura permite que o comportamento de serviços e aplicações e suas interações sejam dinamicamente especificadas e

adaptadas de acordo com regras de alto-nível, baseadas nos requisitos, estados e contexto de execução das aplicações.

4.2 Arquitetura para grades computacionais com gerenciamento autonômico

Este trabalho propõe uma arquitetura de gerência autonômica para ambientes de grades computacionais. Analisando a classificação feita por McCann (MCCANN; HUEBSCHER, 2004), discutida na seção 2.5.6, observa-se que a abordagem que mais se encaixa neste ambiente é a fortemente acoplada. Somente para relembrar, esta abordagem é construída usando agentes inteligentes que, por sua vez, formarão um sistema multi-agente. Nesta abordagem, cada agente tem seus próprios objetivos que são responsáveis por guiar suas ações.

Assim, a idéia é que os elementos autonômicos sejam constituídos por agentes (responsáveis por "sentir"o contexto, tomar as decisões de acordo com políticas de alto nível pré-definidas e atuar sobre ele). A união dos vários elementos autonômicos cria um sistema autonômico, visto também como um sistema multi-agente. A figura 4.1 ilustra esta situação. Deve-se ressaltar que cada elemento autonômico constitui um nó da grade.

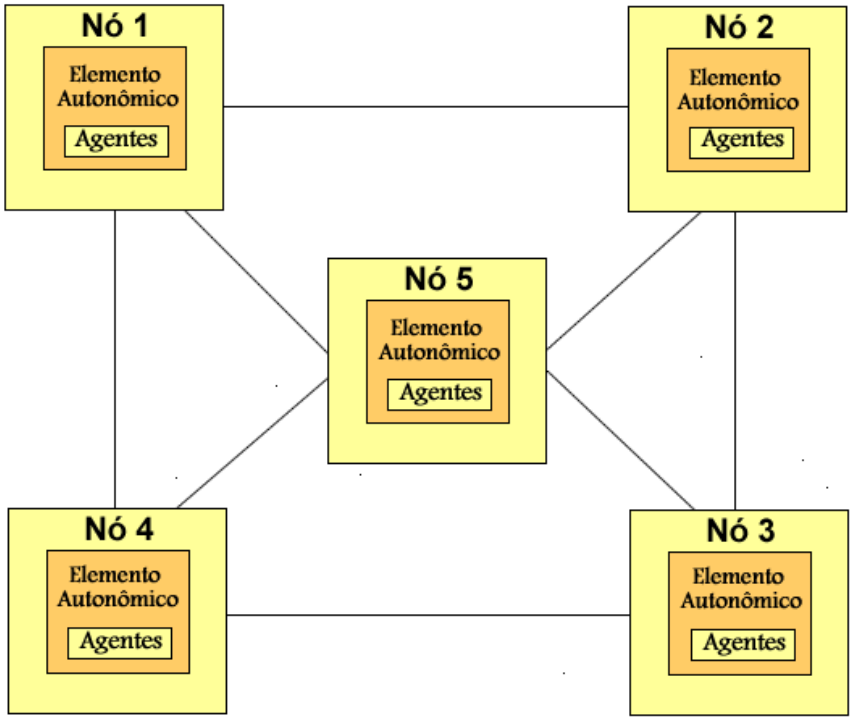


Figura 4.1: Visão geral do sistema autonômico

A unidade funcional da arquitetura é o elemento autonômico (detalhado na seção 2.4.1). Ele é constituído por um gerente autonômico e por elementos gerenciados, como mostrado na figura 4.2. É responsabilidade do gerente autonômico gerenciar os elementos gerenciados (serviços e recursos do nó). Os elementos autonômicos devem possuir sensores e atuadores. Os primeiros são responsáveis por capturar e enviar dados do estado atual dos elementos gerenciados para o gerente. A função dos atuadores é justamente realizar as ações impostas pelo gerente.

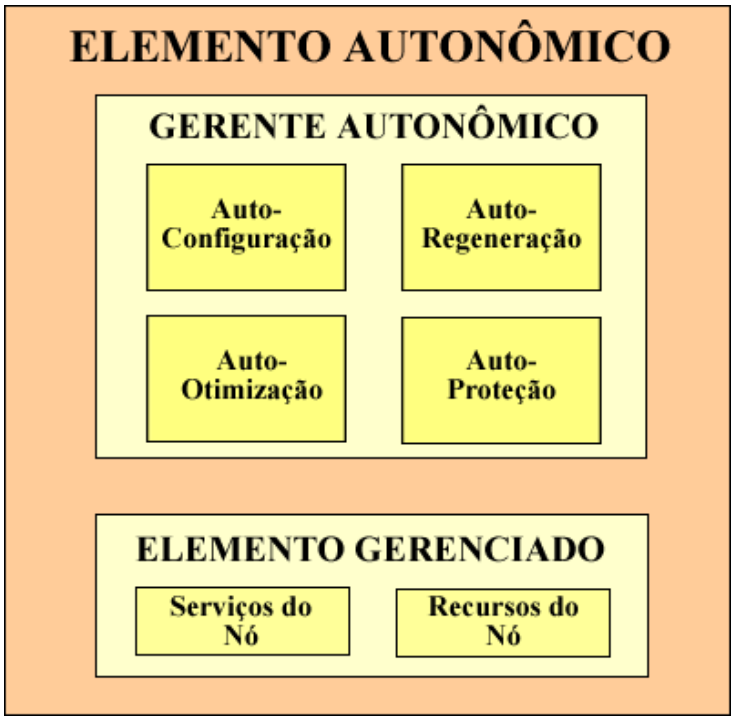


Figura 4.2: Composição de um elemento autonômico

O que faz com que um sistema possa ser chamado de autonômico é a presença do gerente autonômico. Sem ele, não é possível alcançar alto grau de auto-gerenciamento. Através do monitoramento dos elementos gerenciados e de seu ambiente externo, o gerente autonômico é capaz de construir e executar planos de execução baseados na análise das informações enviadas. Portanto, é o gerente autonômico, o responsável por garantir o auto-gerenciamento, alcançado quando todas suas sub-áreas (auto-configuração, auto-regeneração, auto-otimização e auto-proteção) são garantidas também.

Para que o gerente autonômico consiga garantir auto-gerenciamento, este trabalho propõe que ele seja composto por alguns componentes, responsáveis por monitorar os dados enviados pelos elementos gerenciados e outros elementos autonômicos da grade, analisá-los, planejar ações de acordo com seus objetivos e executar estas ações, visando sempre alcançar um alto grau de autonomia. A figura 4.3 mostra a composição do gerente autonômico.

O gerente autonômico é formado por 7 componentes, que juntos proporcionam o compor-

Figura 4.3: Componentes do Gerente Autonômico

tamento auto-gerenciável do sistema. Estes componentes são:

- **Gerência de Serviços (GS)**: responsável por manter uma lista dos serviços disponibilizados pelo nó. Estes serviços estão aptos a aceitar requisições de execuções do próprio nó, assim como de outros elementos da grade. A validade destes serviços deve ser checada periodicamente. Caso alguma falha seja detectada em um serviço oferecido, o nó deve tomar medidas para recriá-lo ou replicá-lo em outro elemento;

- **Redirecionamento de Requisições de Serviços (RRS)**: componente responsável pelo redirecionamento das requisições de serviços enviadas a um nó sobrecarregado. Quando esta situação ocorre, o nó sobrecarregado envia a requisição a um outro nó que está mais ocioso. Caso não exista nenhum outro nó com este serviço, este componente envia um solicitação de replicação para o módulo responsável;

- **Replicação de Serviços**: responsável por replicar serviços oferecidos por um nó em outro elemento da grade. Isto ocorre quando um serviço específico oferecido por um ou mais nós está sobrecarregado. Este componente ao observar que um determinado serviço (disponível em um ou mais nós) está sobrecarregado, escolhe possíveis nós alvos, de acordo com sua ociosidade, para hospedá-lo. Esta medida garante alguns requisitos de qualidade de serviço impostas pelos usuários;

- **Monitoramento de Utilização do Hardware (MUH)**: monitora a utilização e o estado do hardware do nó como velocidade da CPU, disponibilidade do espaço em disco, tamanho da memória e uso das interfaces de rede. Estas informações são armazenadas em *logs* e são utilizadas para saber como está o uso de determinado nó no momento ou determinar um padrão de uso em um intervalo de tempo. As informações de gerência, juntamente com as políticas, formam a base para que as decisões sejam tomadas dentro do elemento;

- **Gerência da Tabela de Rotas (GTR)**: responsável por manter a validade da tabela de rotas. A checagem é feita aproveitando as respostas (ou a falta delas) enviadas pelos elementos remotos quando recebem uma requisição de descoberta de serviços;

- **Tomada de Decisões (TD)**: responsável por tomar decisões baseadas em regras, descritas em políticas definidas. Todas as decisões que exigirem verificação da política do nó são processadas por este componente. É ele que determina se determinado nó tem direito a executar uma tarefa ou se o nó tem interesse e é capaz de receber um serviço replicado;

- **Gerência do Conhecimento (GC)**: componente responsável por gerenciar e manter de forma íntegra as bases de dados do nó. Dentre as bases, encontram-se os dados de gerência armazenados, políticas do nó, descrição dos interesses e capacidades, além de *logs* do elemento. Outra responsabilidade deste componente é tratar os dados para que sejam armazenados de forma correta.

As próximas seções detalharão cada componente da arquitetura.

4.3 Gerência de Serviços - GS

Os nós da grade devem estar aptos a oferecer e requisitar serviços. Existem nós que só requisitam, pois não é de seu interesse oferecer serviços. Porém, para os nós que oferecem, é essencial manter o controle da validade dos serviços disponibilizados.

O controle da validade é feito através do monitoramento das exceções geradas durante os serviços. Se o serviço é executado sem gerar exceções, este componente entende que o serviço está apto a receber novas requisições e respondê-las, pois sua integridade não foi violada.

Quando durante a execução do serviço ocorre uma exceção, o gerente toma conhecimento e reage com o intuito de restabelecer o estado original. A primeira medida é reiniciar o serviço. Se isto acontecer normalmente, sem que novas exceções sejam geradas, o gerente entende que o serviço está íntegro novamente. Caso contrário, o serviço será replicado em outro nó da

grade. As decisões sobre a replicação são de responsabilidade de outros componentes do gerente autonômico.

4.4 Redirecionamento de Requisições de Serviços - RRS

O nó requisitante, quando deseja um serviço, realiza uma busca dentre os elementos da sua tabela de rotas para saber quem tem o serviço e qual é o estado do nó que o detém. Assim, o nó requisitante escolhe o nó mais ocioso.

Apesar disto, existem casos em que um nó sobrecarregado ainda pode receber requisições de execução de serviços de outros nós integrantes da grade. Isto ocorre quando só existe um único nó na grade e este está sobrecarregado, ou quando existe mais de um nó que oferece o serviço, mas todos estão sobrecarregados.

Neste ponto, este componente chama o Replicador de Serviços, outro componente do gerente autonômico. Assim que receber a confirmação que o serviço foi replicado, este componente redireciona a requisição original para o serviço no nó escolhido. Este, por sua vez, executa o serviço e responde para o requisitante. Pode acontecer do componente responsável pela replicação não consiguir replicar o serviço (ex. não há nenhum nó que tenha interesse de hospedar o serviço). Neste caso, mesmo estando sobrecarregado, o nó que recebeu a requisição deve executá-la.

Este componente está diretamente ligado com o Replicador de Serviços, não fazendo sentido pensá-lo de modo separado.

4.5 Replicação de Serviços - RS

Este componente tem por objetivo distribuir o processamento de determinado(s) serviço(s) que possa(m) estar sobrecarregando o processamento em um nó ou um grupo de nós. É muito útil para grades nas quais a demanda por serviços fornecidos por poucos nós torna-se muito grande, pois se nenhuma ação for tomada nesta situação e a demanda continuasse alta ou crescente, a grade certamente viria a passar por problemas de performance e seriam criados gargalos. Por isto, esta funcionalidade se encaixa no conceito de otimização no auto-gerenciamento.

A ação de replicar um serviço consiste basicamente em transferir, de um nó para outro, a implementação do código fonte do serviço na linguagem onde o ambiente de grade foi definido. Para exemplificar, dentre as formas de se fazer isto, cita-se a transferência de arquivo compilado

executável ou de código fonte para compilação no nó destino. Então, completando o processo, o nó recebendo a replicação inclui o fonte recebido com a implementação do serviço em sua biblioteca e inclui em sua lista de serviços oferecidos o serviço replicado, disponibilizando-o à grade.

A figura 4.4 mostra a representação de dois momentos de uma grade para demonstrar o que é a replicação de um serviço. No início, somente o nó 2 possui disponível determinado serviço, representado pela cor azul. Em determinado momento, a demanda por solicitações de serviços no nó 2 faz com que diminua muito a capacidade de recursos nele e então o serviço é replicado para outro nó com maior disponibilidade naquele momento, nó 4, dividindo o processamento das demandas entre estes dois nós.

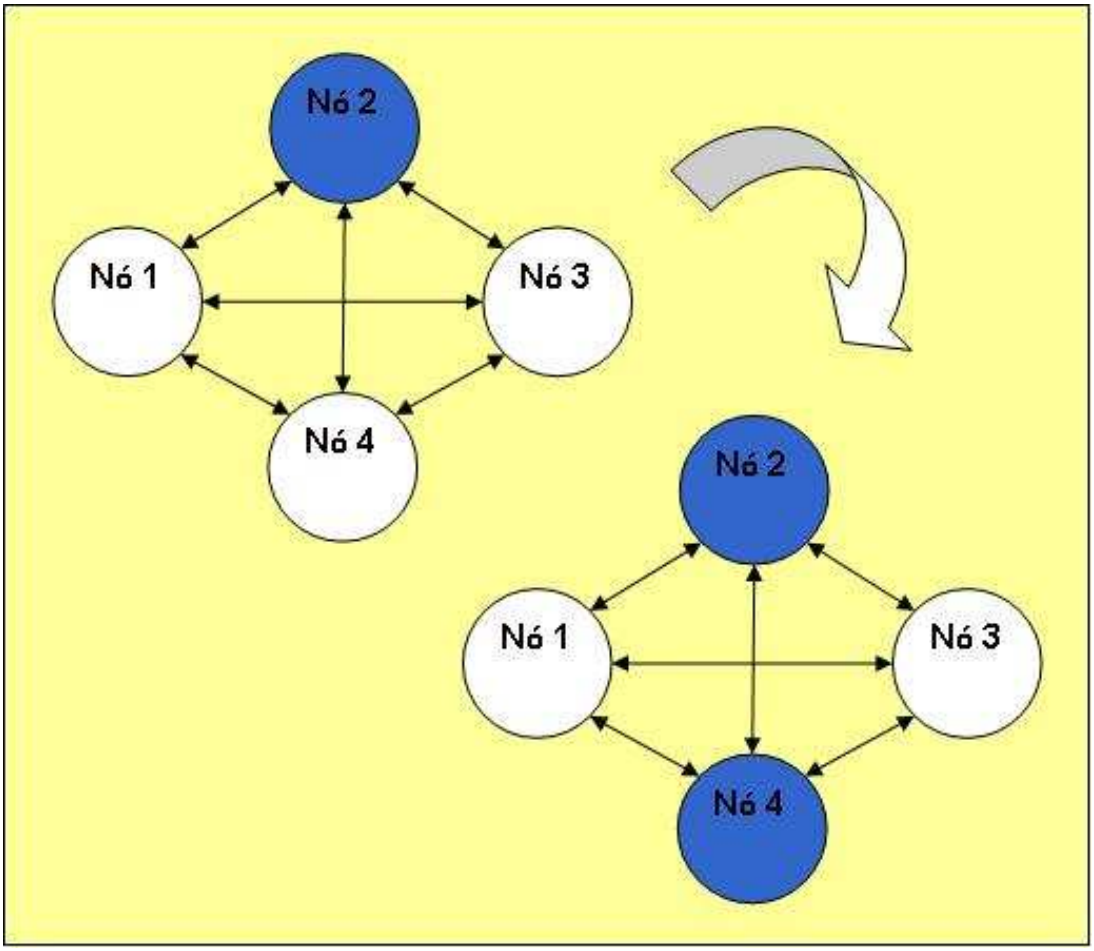


Figura 4.4: Replicação de serviço do nó 2 para o nó 4.

Os nós de uma grade possuem uma lista de serviços básicos e conhecidos por todos que garantem o funcionamento do ambiente. Para realizar a replicação de serviços são necessários dois novos serviços básicos:

- Replicar Serviço: utilizado pelo nó para replicar um serviço da grade quando solicitado.

Este serviço tem como parâmetros de entrada o nome do serviço a ser replicado e o nó destinatário da replicação. Ao executar este serviço o executante envia uma tarefa com as informações necessárias para a implementação do serviço a ser replicado ao nó destinatário solicitando que este execute o serviço;

- Receber Serviço: utilizado pelo nó sem o serviço para que possa receber uma replicação. Tendo como parâmetros de entrada o nome e as informações da implementação do serviço replicado (e.g. código fonte, nome do objeto, localização do executável na rede, etc.). Este serviço deve adicionar a implementação em sua biblioteca e adicionar o serviço na lista de serviços disponibilizados pelo nó.

Sempre que houver uma replicação estes dois serviços são executados nesta ordem, "replicar serviço"e "receber serviço", respectivamente, pelo nó replicador e nó destino do serviço. Estas situações de replicação são abordadas no tópico a seguir que define uma política de roteamento de serviços.

4.5.1 Roteamento de Serviços

Tendo-se definido o que é a replicação e como ela acontece, deve-se determinar quando ela acontece. A intenção não é que serviços sejam replicados à vontade na grade e que todos os nós forneçam todos os serviços. Este trabalho tem uma proposta de roteamento a ser executado por qualquer nó, sempre que necessitar solicitar um serviço qualquer da grade, definindo as situações em que pode ocorrer replicação.

Definições e Métricas

Para a definição do roteamento proposto é necessário entender algumas definições e métricas dadas aos nós:

- Percentual de recurso livre: é a relação entre recursos do sistema livre e recursos totais de um nó, ponderando 70% ao processador e 30% à memória, seguindo a seguinte fórmula:

$$\frac{(\text{Percentual\_Processador\_Livre} \times 0{,}7 + \text{Memória\_Livre} \times 0{,}3) \times 100}{(\text{Percentual\_Processador\_Total} \times 0{,}7 + \text{Memória\_Total} \times 0{,}3)}$$

- Nível de recursos: os recursos de uma grade normalmente são heterogêneos, de diferentes tipos e com capacidade de recursos diferentes, portanto não se pode compará-los como se fossem todos iguais para fins de performance. Para resolver isto, é possível criar uma escala de níveis com a granularidade que for mais conveniente. Nesta escala, o nível zero é utilizado para os nós que nunca fornecerão serviços não-básicos à grade (e.g. sensores). O nível 1 é utilizado para os nós que fornecem serviços com menores capacidades de recursos e à medida que a capacidade de recursos melhora, os nós avançam níveis na escala até o último nível que corresponde aos nós com melhores capacidades;

- Recurso disponível: é o "nível de recursos"de um nó multiplicado pelo seu "percentual de recursos livre". Este valor nivela os diferentes tipos de recursos da grade (nós) para um mesmo patamar de acordo com a escala de níveis de recursos criada, sendo calculada com a fórmula:

  ```
  Nível_de_Recursos x Percentual_Recursos_Livre
  ```

- Nó sobrecarregado: considera-se que um nó está sobrecarregado quando o seu "recurso disponível"no momento é menor do que N, sendo N um valor definido pela política da grade. N representa o limite de "percentual de recursos livre", no qual um nó com "nível de recursos"igual a 1 ainda não é considerado sobrecarregado. Se, por exemplo, N for igual a 90, isto significa que um nó com "nível de recursos"igual a 1 passa a ser considerado sobrecarregado quando seu "percentual de recursos livre"for menor do que 90.

A tabela 4.1 mostra uma comparação com o percentual de recursos livre mínimo para que um nó não seja considerado sobrecarregado em cada nível de recursos, tendo uma granularidade de níveis de recursos que vai de zero a cinco e tomando-se N (limite de recursos livre do nível 1) igual a 90.

Utilizando esta tabela, nós com níveis de 1 a 5 com os seus respectivos percentuais de recurso livre limite disponível estarão niveladas com o mesmo valor de recurso disponível (produto entre nível e percentual de recursos livres). Portanto, para um nó do nível 5 com 18% de recursos livre, um nó do nível 2 precisará ter 45% de recursos livre para considerarmos que ambas tem o mesmo recurso disponível, mostrando que o nó do nível 5 é 2,5 vezes mais rica em recursos que o nó do nível 2.

Tabela 4.1: Tabela sobre os percentuais de recursos livre limite

Nível	Percentual de Recursos Livre Limite
0	*
1	90%
2	45%
3	30%
4	22,5%
5	18%

Algoritmo de Roteamento de Serviços

Ao solicitar a execução de um serviço na grade, o nó solicitante não sabe quem executará a tarefa, portanto não define qual elemento deve recebê-la. O algoritmo de roteamento encarrega-se de pesquisar o nó mais adequado para executá-la no momento e, se for necessário, para isto fará uma replicação de serviço.

O algoritmo inicia seu processo fazendo uma varredura por todos os seus nós adjacentes, os quais o solicitante possui rota, e monta uma tabela com as informações retornadas. Para consultar os nós diretamente conectados em busca das informações necessárias inclui-se um novo serviço básico aos nós:

- Pesquisar Serviço (Replicação): quando solicitado por este serviço o nó responde se possui ou não o serviço, qual o seu recurso disponível atual e se aceita receber replicação de serviço (política do nó).

O objetivo é encontrar um nó, incluindo o próprio solicitante, que seja mais adequado a executar o serviço. Os critérios considerados para esta classificação, em ordem decrescente de importância, são:

- não estar sobrecarregado;
- possuir o serviço;
- ter mais recurso disponível;
- aceitar replicação, sendo este um critério obrigatório.

As figuras 4.5 e 4.6 apresentam os fluxogramas que utilizam estes critérios para definir qual nó deve executar o serviço e, se for o caso, entre quais nós deve ocorrer replicação. Quadros verdes significam respostas afirmativas e quadros vermelhos respostas negativas.

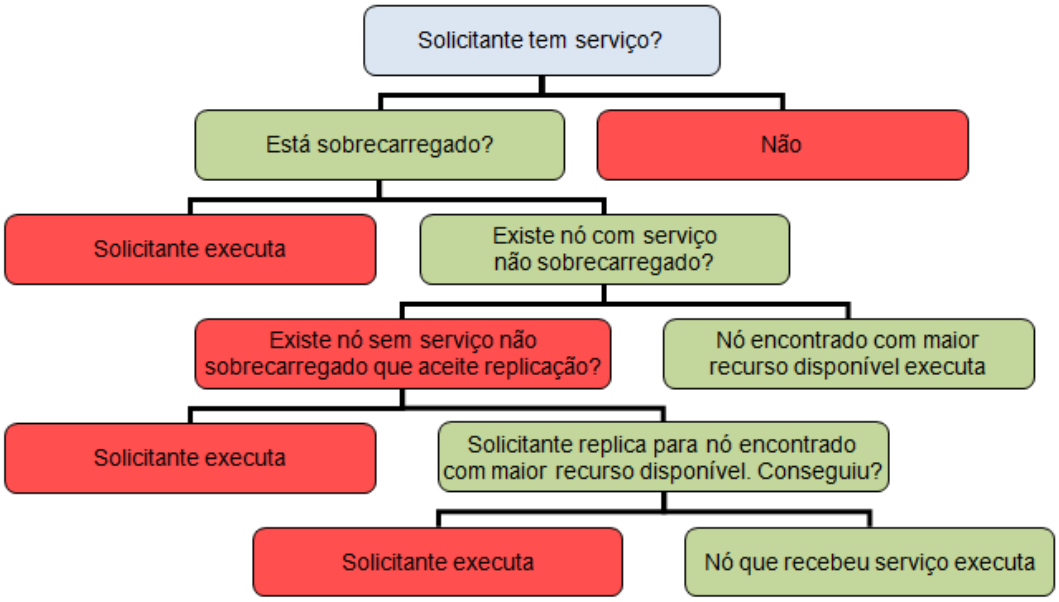


Figura 4.5: Fluxograma da replicação quando o solicitante tem o serviço

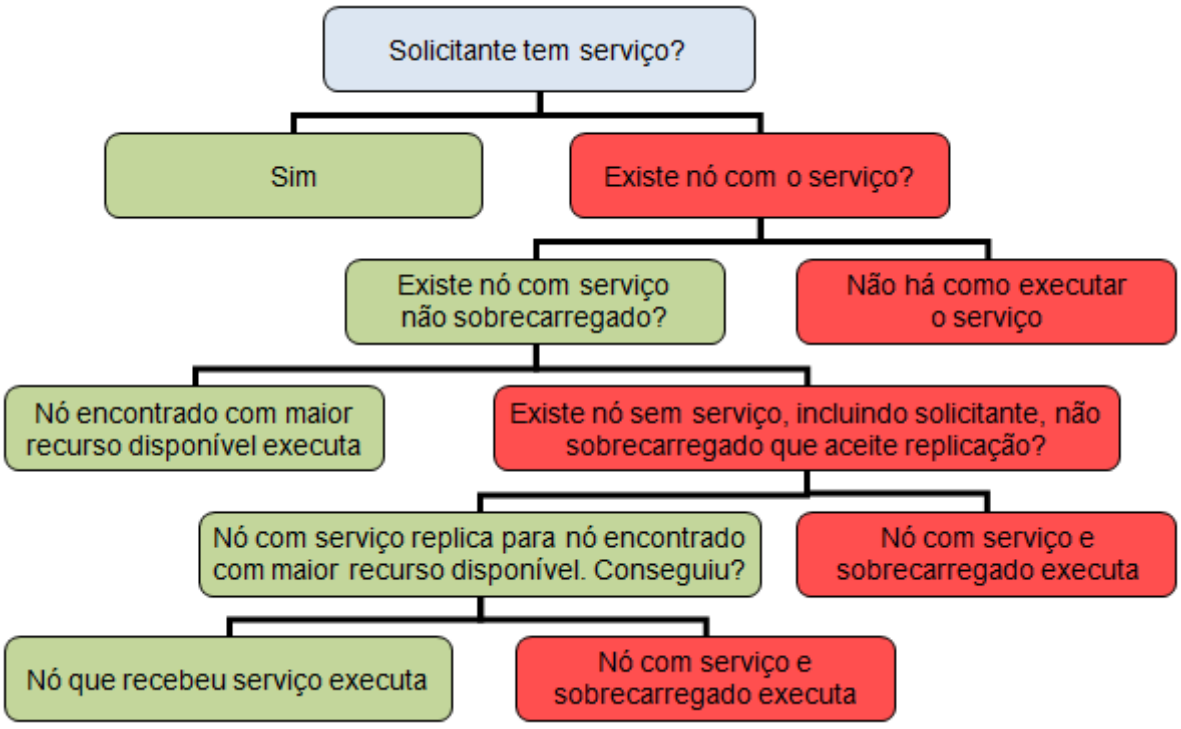


Figura 4.6: Fluxograma da replicação quando o solicitante não possui o serviço

Como se pode perceber, a replicação só ocorre quando todos os nós que possuem o serviço estão sobrecarregados e existe algum nó sem serviço, não sobrecarregado e que aceite receber a replicação. Pode acontecer do nó que replica o serviço ser o próprio solicitante. Isto ocorre caso ele aceite replicação de serviços e seja o elemento com maior disponibilidade de recursos.

4.6 Monitoramento de Utilização do Hardware - MUH

Com o intuito de gerenciar os elemento da grade, cada nó deve possuir um componente responsável por realizar a coleta das informações do hardware e guardar em um banco de dados, formando um *log* do elemento. Neste trabalho, este componente é chamado de Sensor, discutido na seção 4.6.1.

Porém, a grade de computadores apresenta um conjunto muito heterogêneo com vários sistemas operacionais, hardwares, etc. Portanto, tem-se um cenário com vários contextos diferentes, sendo que cada contexto deve ser gerenciado de modo específico, ou seja, executando funções internas específicas. Assim, conclui-se que o monitoramento é totalmente dependente do tipo do sistema (BALEN; FRANKE; WESTPHALL, 2008).

Este monitoramento é feito seguindo três passos:

1. **Coleta dos Dados:** Neste passo, este componente irá executar funções internas, que revela o estado do contexto de execução do elemento. Após cada execução de método, os dados retornados são repassados ao seu respectivo *parser* (cada comando tem o seu *parser* específico). Mais informações sobre como ocorre a coleta do contexto do nó são dadas na seção 4.6.2;

2. ***Parsers*:** Este passo tem a função de filtrar dos dados coletados o que é realmente importante. Ao executar um comando, por exemplo, são retornados dados relevantes e irrelevantes. Cabe ao *parser* analisar isto. Após realizada a filtragem, os dados são repassados ao passo seguinte, responsável por criar as *XMLTree*. Os *parsers* são discutidos com mais detalhes na seção 4.6.5;

3. **Criação do *XMLTree* e armazenamento na base de dados:** Para que os dados tenham um formato padrão, os dados relevantes oriundos de todos os *parsers* de cada comando executado são formatados em forma de árvore XML. Isto facilita tanto a compreensão dos dados como também a fácil manipulação no momento do seu armazenamento no banco de dados.

Há mais um passo possível, no qual um usuário interessado requer um relatório sobre o elemento. Isto é feito realizando uma requisição ao serviço "*viewer*", responsável pela geração dos gráficos e relatórios.

4.6.1 Sensores

Para que os dados possam ser coletados, dentro de cada nó deve haver um coletor. O Grid-M fornece um modo simples de construção dos coletores, apenas utilizando a sua API. O exemplo abaixo mostra como criar um sensor.

```
public class ExampleSensor extends Sensor {
    ...
    public XMLTree _collect(Node node){
        /*Aqui informamos o que queremos que seja coletado a cada
        intervalo de tempo*/
    }
    ...
}
```

Para criar os sensores portanto, basta estender a classe abstrata *Sensor* e especificar o que deve ser coletado, ou seja, quais *scripts* ou comandos devem ser executados, dentro do método *_collect*.

4.6.2 Coleta do Contexto

Contexto é uma das palavras chaves para o monitoramento. A grade é vista de maneira homogênea. Mas não é homogênea a maneira de coletar os dados dos dispositivos que a compõe. Cada dispositivo apresenta maneiras diferentes de acesso aos seus dados de gerenciamento. Por exemplo, no Windows, o modo de descobrir dados sobre a interface de rede é através do comando *ipconfig*. No Linux, tal comando não existe, mas sim, *ifconfig*, que retorna dados quase similares ao comando do Windows.

Deve-se ressaltar que cada nó tem um contexto. Por isso cada contexto será tratado de forma específica. Portanto, para cada dispositivo da grade deverão ser criados modos diferentes de coleta. Métodos que podem ser executados em mais de um dispositivo são reaproveitados. Por exemplo, o modo de coleta das conexões abertas de um nó é a mesma tanto no Linux como no Windows, através do comando *netstat*.

Para realizar a coleta, os sensores utilizam basicamente dois métodos de coleta, os *scripts* escritos em *Virtual Basic Script* para ambientes Windows, e a execução de linhas de comandos para todos os ambientes. Cada método será detalhado nas seções seguintes.

4.6.3 Execução de Linhas de Comando

Os coletores de linha de comando extraem as informações de gerenciamento através da execução de utilitários de linha de comando disponíveis nos diversos sistemas operacionais, no quais podem ser executados.

No ambiente UNIX, os utilitários de linha de comando podem consistir de comandos como *netstat*, *top*, *df*, *free*, *ifconfig*, dentre outros. A execução de comandos retorna um fluxo de *bytes* do qual são extraídos os dados relevantes, através dos *parsers*.

Estes coletores fornecem as funcionalidades necessárias para a construção de agentes de coleta com a flexibilidade e as características requeridas por uma aplicação de gerenciamento de sistemas (ASSUNCAO, 2004). Exemplo de coletor de linha de comando é o apresentado na figura 4.7, no qual é executado o comando *ifconfig* no ambiente UNIX.

```
eth0       Encapsulamento do Link: Ethernet  Endereço de HW
00:08:54:12:51:C2
          inet end.: 10.1.1.3  Bcast:10.255.255.255  Masc:255.0.0.0
          endereço inet6: fe80::208:54ff:fe12:51c2/64 Escopo:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Métrica:1
          RX packets:106856 errors:0 dropped:0 overruns:0 frame:0
          TX packets:65122 errors:0 dropped:0 overruns:0 carrier:0
          colisões:0 txqueuelen:1000
          RX bytes:148931656 (142.0 MiB)  TX bytes:7414102 (7.0 MiB)
          IRQ:17 Endereço de E/S:0x9800
```

Figura 4.7: Coletor de linha de comando no UNIX

No caso de dispositivos que apresentam interfaces seriais, como o WRT, WLAN e outros poderiam-se executar comandos desse tipo, como ilustrado na figura 4.8.

4.6.4 Coletores WMI

Os coletores WMI são extensões dos agentes coletores de linha de comando. Estes coletores extraem dados necessários às atividades de gerenciamento usando o pacote de instrumentação WMI proporcionado pela plataforma Microsoft Windows (MICROSOFT, 2003).

```
cat /proc/tty/driver/serial
serinfo:1.0 driver:5.05c revision:2001-07-08
0:  uart:16550A port:B8000300 irq:3 baud:113636 tx:326 rx:0
RTS|CTS|DTR|DSR|CD
1:  uart:16550A port:B8000400 irq:0 tx:0 rx:0 CTS|DSR|CD
```

Figura 4.8: Coletor de interfaces seriais

Para isso estes coletores executam *scripts*, desenvolvidos em Visual Basic Script, que são responsáveis por interagir com o gerenciador de objetos do WMI e por recuperar estes dados. Quando passados como parâmetros de utilitários existentes para a execução destes *scripts*, eles apresentam o resultado na console do sistema operacional, como se fosse executado um comando do sistema (ASSUNCAO, 2004).

Estes resultados são usados pelos sensores que realizam o mesmo processo que um coletor de linha de comando para extrair estas informações. Em seguida, o coletor efetua a extração das informações relevantes do resultado da mesma forma que é feito com os agentes de linha de comando. A figura 4.9 ilustra de maneira abstrata como este processo acontece, no qual, primeiramente, o *script* é executado (1), posteriormente, o resultado desta execução é tratado (2) e os dados são formatados em XML (3) (ASSUNCAO, 2004).

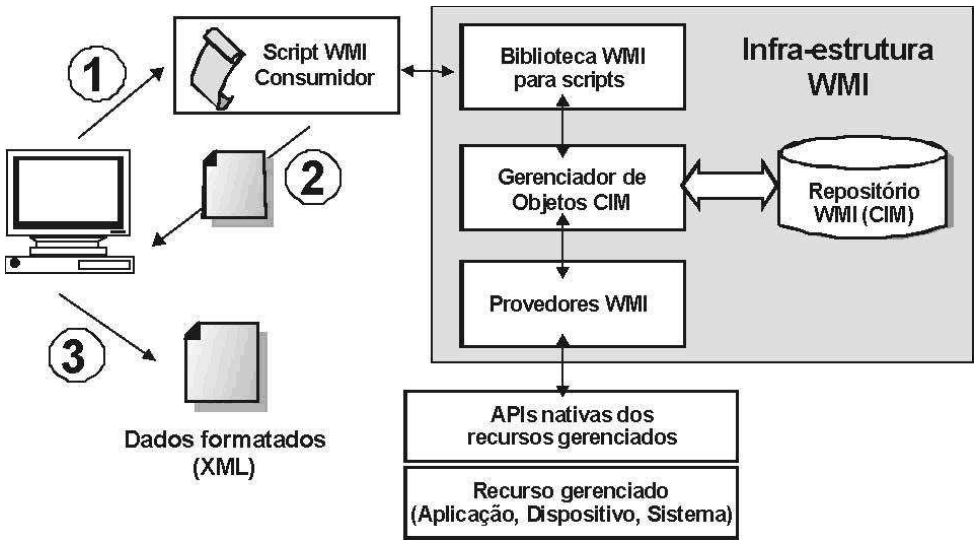


Figura 4.9: Funcionalidade de um coletor WMI (MICROSOFT, 2003)

As regras para funcionamento dos coletores WMI são semelhantes as usadas pelos coletores de linha de comando. Neste trabalho, foram desenvolvidos *scripts* para a coleta de informações de configuração e desempenho a respeito de processadores, memória, periféricos e outros. Exemplo da saída de um coletor de informações de desempenho do processador é apresentado

na figura 4.10.

```
Address Width: 32 Architecture: 0 Availability: 3 CPU Status: 1
Current Clock Speed: 1913 Data Width: 32 Description: x86
Family6Model 10 Stepping 0 Device ID: CPU0 External Clock: 166
Family: 29 L2 Cache Size: 512 L2 Cache Speed: 637 Level: 6 Load
Percentage: 0 Manufacturer: AuthenticAMD Maximum Clock Speed: 1913
Name: AMD Athlon(tm) XP 2600+ PNP Device ID: Processor ID:
0383FBFF000006A0 Processor Type: 3 Revision: 2560 Role: CPU Socket
Designation: Socket A Status Information: 3 Stepping: 0 Unique Id:
Upgrade Method: 4 Version: Modelo 10, Nível 0 Voltage Caps:
```

Figura 4.10: Coletor de informações do processador

Outro exemplo, é o coletor de informações dos discos rígidos disponíveis no nó, mostrado na figura 4.11.

```
DeviceID:  C:       Free Disk Space: 27728949248
DeviceID:  D:       Free Disk Space: 6195707904
```

Figura 4.11: Coletor de informações dos discos rígidos

4.6.5 *Parser*

Os *parsers* tem a função de filtrar os dados relevantes para o gerenciamento. Exemplo disto é quando se executa um *script* que retorne dados referentes ao processador. Após a execução, são retornados vários dados, como tamanho das *caches*, a versão, etc, mas pode ser que o único dado interessante seja o percentual de uso da CPU. Neste caso, o *parser* iria procurar dentre todos os dados retornados e selecionar somente o percentual de uso da CPU.

Cada comando de linha ou *script* executado deve passar pelo seu respectivo *parser*. Há um *parser* para cada contexto, ou seja, o *parser* depende do comando e do sistema operacional do elemento coletado. Por exemplo, o *parser* do *script* para retornar dados da CPU, poderia ser de um tipo para o Windows XP e outro para Windows 98, caso a saída dos dois fossem diferentes.

Para a criação do *parser*, deve-se implementar a classe Parser:

```
public interface Parser{
    ...
    public XMLTree parse(String output);
    ...
}
```

4.6.6 Criação das *XMLTree*

As *XMLTree* como o próprio nome diz, são árvores XML. Elas facilitam tanto a compreensão para uma pessoa que abra o arquivo XML para observar as informações, pois apresentam dados estruturados, como também é um padrão e sendo assim, facilita o seu armazenamento, já que todos apresentam o mesmo formato.

Os dados coletados pelos sensores são enviados ao componente responsável pela gerência do conhecimento. A figura 4.12 exibe um exemplo dos dados coletados do contexto do nó 1 no formato de árvore XML.

```
<Node1>
<Net>
<NetworkTrafficReceived>163076Kb</NetworkTrafficReceived>
<NetworkTrafficSent>66739 Kb</NetworkTrafficSent>
<NetworkErrorsReceived>0 Kb</NetworkErrorsReceived>
<NetworkErrorsSent>0 Kb</NetworkErrorsSent>
</Net>
<Memory>
<FreePhisicalMemory>30 MB</FreePhisicalMemory>
<FreeVirtualMemory>2003 MB</FreeVirtualMemory>
</Memory>
</Node1>
```

Figura 4.12: Dados coletados do nó 1 formatados em XML

4.6.7 Armazenamento na base da dados

Com o intuito de gerenciar os elementos autonômicos, os dados devem ser armazenados, criando históricos de cada dispositivo da grade. A cada intervalo de tempo, os nós enviam seus contextos para o componente Gerência do Conhecimento (seção 4.9), responsável pelo armazenamento. Assim, é responsabilidade deste componente garantir que os dados serão armazenados de forma otimizada e com o formato (sintaxe) correto.

4.7 Gerência da Tabela de Rotas - GTR

O nós em uma grade podem apresentar um comportamento muito volátil, podendo entrar e sair da grade sem aviso prévio. Em se tratando de grades que permitem a conexão de ambientes móveis, esta situação se agrava ainda mais. Com as constantes entradas e saídas de dispositivos,

os elementos ficam, com o decorrer do tempo, com a sua tabela de rotas inválidas, sem refletir a veracidade do ambiente.

Com o intuito de tratar este problema, este componente é responsável por manter a validade da tabela de roteamento do nó. Os tipos de roteamento utilizados neste trabalho serão discutidos na seção 4.10. É através desta tabela que o gerente autonômico detecta se um elemento remoto pertencente a sua tabela está indisponível.

A forma mais fácil e intuitiva de realizar o monitoramento é mandando mensagens de *keep-alive* em intervalos fixos de tempo para todos os elementos da tabela. Porém, se a tabela de um elemento for grande (ex. 100 nós), ele irá desperdiçar tempo, processamento e largura de banda só com a gerência do envio e recebimento destas mensagens.

Como um dos objetivos deste trabalho é fornecer um ambiente no qual dispositivos com pouco poder de processamento estão presentes, não faz sentido neste contexto pensar em usar mecanismos de *polling* [1] para o monitoramento.

Assim, a verificação é realizada de outra forma. Quando um nó deseja descobrir um elemento autonômico que forneça determinado serviço, ele faz uma requisição de descoberta (*discover service*) aos nós que estão na sua tabela de roteamento. A figura 4.13 ilustra esta situação.

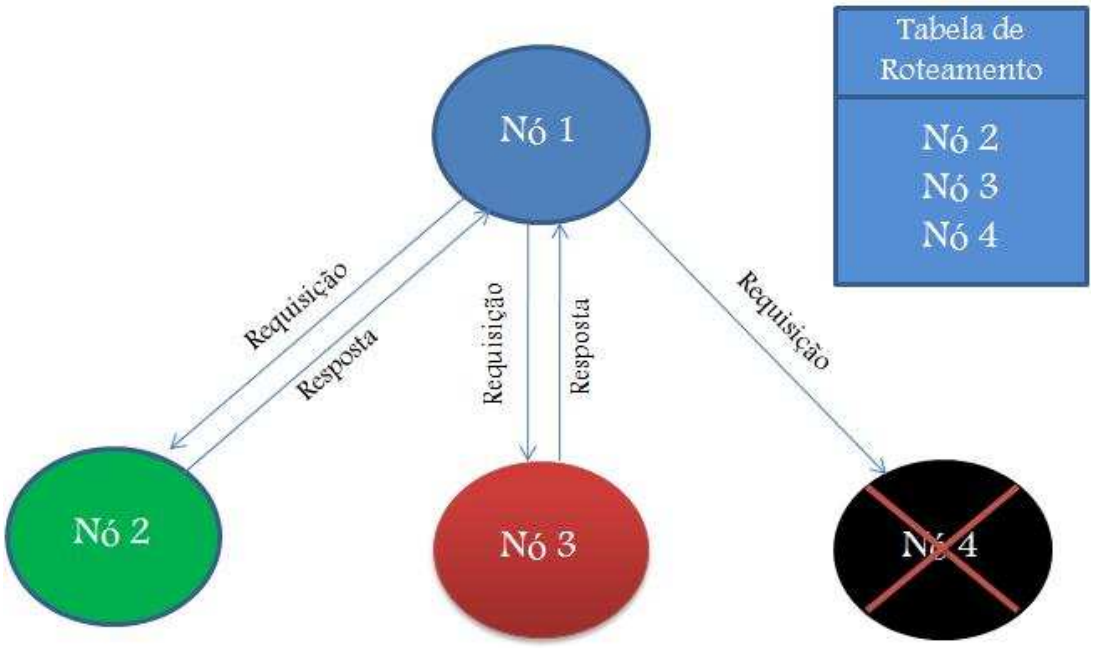


Figura 4.13: Execução de uma requisição de descoberta

No exemplo, o nó 1 quer descobrir quem tem um determinado serviço. Os elementos 2, 3 e

[1] Sistemas de *polling* são aqueles nos quais um dispositivo requisita em intervalos de tempo fixo o estado de um dispositivo remoto.

4 estão na sua tabela. Assim, o nó 1 envia as requisições para os todos os elementos da tabela. O nó 2 responde de forma positiva. O nó 3 responde que não possui o serviço. Já o nó 4 não responde. Isto significa que, aparentemente, o elemento 4 está indisponível. Diante disto, o gerente autonômico decide colocá-lo em quarentena e uma nova requisição é enviada depois de um certo tempo (definido no arquivo de política do elemento), como mostrado na figura 4.14.

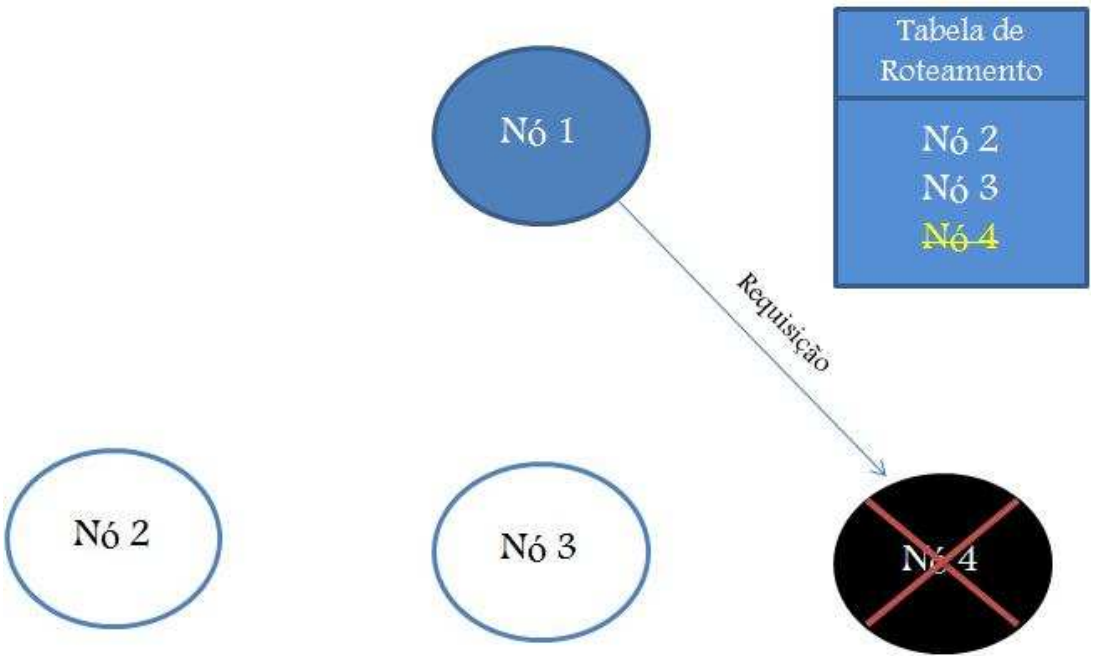


Figura 4.14: Reenvio de uma requisição de descoberta

Caso o nó 4, que estava em quarentena, responda a requisição, este componente entende que o elemento está funcionando normalmente e retira-o do estado de observação. Caso contrário, o nó 1 retira da sua tabela de roteamento o nó 4 e passa esta informação para os outros elementos da sua tabela (nós 2 e 3), pois esta pode interessá-los.

Portanto, através de requisições de descoberta de serviços, este componente garante a validade da tabela de rotas. Assim, é possível garantir validade sem utilizar técnicas de *polling* para o monitoramento. O único ponto negativo é que a verificação fica limitada a uma solicitação de serviço. Porém, como tudo é orientado a serviços, a probabilidade de detecção de indisponibilidades torna-se grande.

4.8 Tomada de Decisões - TD

No momento de requisitar um serviço e/ou pedir replicação, o elemento requisitante não sabe se o nó receptor tem o interesse ou meios de executá-lo. Pode ser que um serviço só

esteja ativo em determinado horário ou só para alguns usuário específicos. Assim, usuários sem permissão ou requisições que chegarem fora da faixa de tempo disponível serão barrados.

Além disto, o nó requerente pode desejar uma resposta em um tempo máximo (ex. 2s para obter a resposta). Porém, o elemento que detém o serviço sabe, através das suas estatísticas, que este serviço demora em média X segundos (ex. 5s) para executar. Como não poderá responder no tempo desejado, ele rejeita a requisição.

Resumindo, as decisões são tomadas levando em consideração a combinação entre a requisição do elemento requisitante e as políticas (discutidas em detalhes na seção 4.9) do elemento receptor. Este componente é, justamente, o responsável por realizar esta combinação e determinar quais serão as próximas ações.

Com o intuito de facilitar a compreensão deste componente e torná-lo mais modular, ele foi dividido em sub-componentes, ilustrados na figura 4.15. É através da interação desses sub-componentes que o gerente autonômico toma suas decisões. A arquitetura deste componente foi baseada na linguagem XACML (OASIS, 2005), porém com simplificações se comparada com esta última.

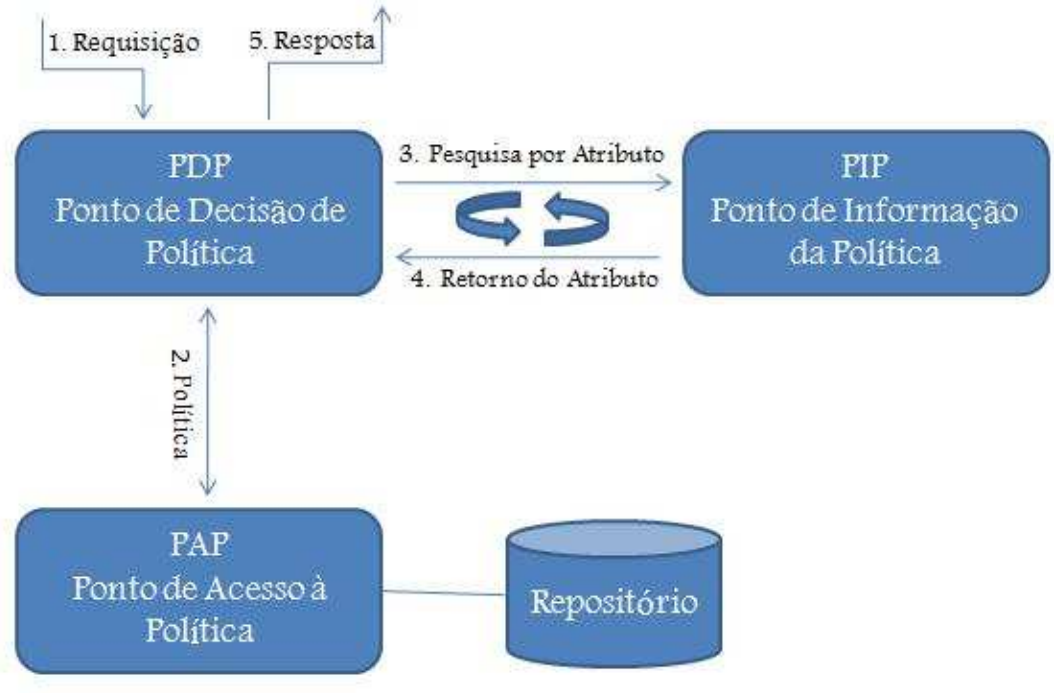


Figura 4.15: Sub-componentes responsáveis pelas decisões

Os sub-componentes (figura 4.15) são:

- Ponto de Decisão de Política (PDP): a requisição, em forma de tarefa, chega a este sub-componente (passo 1). Por sua vez, ele avalia a requisição e envia a resposta (passo

5). A resposta pode permitir ou negar o acesso. A decisão é tomada depois de avaliar as políticas relevantes e suas respectivas regras. Mas para ter acesso às políticas do elemento, ele necessita de um outro sub-componente, o PAP;

- Ponto de Acesso à Política (PAP): este sub-componente é responsável por criar políticas e configurá-las, além de torná-las disponíveis para o PDP (passo 2). Na verdade, este componente gerencia o repositório de políticas do elemento, mas se limita apenas a acessar as políticas, não a tratá-las;

- Ponto de Informação da Política (PIP): responsável pelo acesso aos atributos da política. Depois de ter acesso às política do nó, o PDP deve ter acesso aos atributos da mesma (passos 3 e 4). Isto é feito através do PIP.

4.9 Gerência do Conhecimento - GC

A computação autonômica baseia suas decisões sobre políticas de alto nível. Elas são definidas por administradores, que conhecem a forma de execução do sistema. Assim, elimina-se intervenções diretas de administradores no ambiente, limitando suas atividades em criar e otimizar as políticas definidas.

Porém, para que o Gerente Autonômico possa tomar decisões baseadas nessas políticas, é necessário que ele tenha acesso as mesmas, assim como aos dados de gerência armazenados. Assim, as políticas e os dados de gerência são a fonte de conhecimento do elemento, e todas as decisões são baseadas neles.

O acesso à base de conhecimento do elemento deve ser feito de maneira correta e otimizada. É de responsabilidade deste componente garanti-lo. Deve-se ressaltar, porém, que este componente trata da forma (sintaxe) como estes dados são armazenados, não se preocupando em fazer inferências semânticas sobre eles, já que isto é função do componente Tomada de Decisões (seção 4.8).

As políticas são armazenadas no nó na forma de arquivos XML. Há três tipos de políticas que podem ser definidas: *sistema, serviços, serviço-<nome-do-serviço>*. A primeira refere-se ao sistema como um todo, contendo suas capacidades, condições e disponibilidade. Deve existir no mínimo uma política de sistema. Caso haja mais que uma, a política mais recente será respeitada. A figura 4.16 exibe uma política de sistema.

A política de sistema define as características do mesmo. Levando em consideração o exemplo anterior, vê-se que o sistema permite replicação e colaboração, e que não possui restrição

```
<politica>
<nome>sistema1</nome>
<descricao>Define as características do sistema</descricao>
<alvo>sistema</alvo>
<capacidade>
    <colaboracao>sim<\colaboracao>
    <replicacao>sim<\replicacao>
</capacidade>
<condicao>
    <valor_N>90</valorN>
    <nivel_recurso>5</nivel_recurso>
    <tempo_espera>60<tempo_espera>
</condicao>
<disponibilidade>
    <horario>qualquer</horario>
    <data>qualquer</data>
</disponibilidade>
</politica>
```

Figura 4.16: Exemplo de política de sistema

de disponibilidade. Além disso, fixa alguns valores (na forma de condições) que servem de parâmetro para o Gerente Autonômico.

Outro tipo de política é a de serviços. Esta política é adotada como política padrão para os serviços oferecidos pelos nós. Caso um serviço tenha características diferentes do padrão, pode-se criar uma política específica para este serviço (política serviço-<nome-do-serviço>). Deve haver no mínimo uma política de serviço. Um exemplo deste tipo de política é exibida na figura 4.17.

```
<politica>
  <nome>serviços1</nome>
  <descricao>Define as restrições dos serviços oferecidos</descricao>
  <alvo>serviços</alvo>
  <condicao>
    <max_requicoes>100</max_requicoes>
  </condicao>
  <disponibilidade>
    <horario>08:00-20:00</horario>
    <data>10/01/2009-10/03/2009</data>
  </disponibilidade>
</politica>
```

Figura 4.17: Exemplo de política de serviços

De acordo com a política acima, o número máximo de requisições que um serviço pode aceitar em um dia é 100 e a disponibilidade dos serviços será das 08h até às 20h entre os dias 10 de janeiro e 10 de março de 2009. Fora destes dias, por padrão, todas as requisições serão negadas, a não ser que exista uma política específica para um serviço que especifique outra disponibilidade.

A política deve ser construída por *tags* bem definidas, de acordo com a seguinte gramática:

```
<nome>::sistema{0-9}|serviços{0-9}|serviço-{nome-do-serviço}
<descricao>::{seqüencia de caracteres}
<alvo>::sistema|serviços|serviço-{nome-do-serviço}
<capacidade>::<colaboracao>|<replicacao>
<colaboracao>::sim|não
<replicacao>::sim|não
<condicao>::<valor_N>|<nivel_recurso>|<tempo_espera>|<max_requisicoes>|
            <max_tempo_execucao>
<valor_N>::0-100
<nivel_recurso>::0-5
<tempo_espera>::1-120|qualquer
<max_requisicoes>::1-1000000|qualquer
<max_tempo_execucao>::1-60000|qualquer
<disponibilidade>::<horario><data>
<horario>::horario-horario|qualquer
<data>::data-data|qualquer
```

Respeitando esta gramática, pode-se criar várias políticas com características diferentes entre elas. Após a política ser criada e sua sintaxe validada, este componente em conjunto com o módulo do *middleware* responsável pelo armazenamento, guarda e gerencia o acesso a mesma. Como dito anteriormente, as políticas são armazenados na forma de arquivos XML.

Juntamente com as políticas, os dados de gerência coletados formam a base de conhecimento do elemento. Os dados chegam em intervalos fixos, definidos pelo MUH (seção 4.6), no formato de XMLTree. Ao chegarem, os dados são armazenados em um banco de dados, permitindo um acesso otimizado a eles.

4.10 Roteamento entre os nós

O ambiente formado por dispositivos móveis está sujeito a grandes mudanças, devido às constantes entradas e saídas dos mesmos. Manter a tabela de roteamento consistente é fundamental para a inter-conexão entre os nós. O componente de Gerência da Tabela de Rotas tem função importante na detecção de inconsistências, mas não tem função direta na manipulação da tabela de roteamento. Isto é feito pelo algoritmo de roteamento da grade.

Neste sentido, o sistema proposto neste trabalho implementa dois algoritmos de roteamento: um deles é baseado na interconexão direta com o nó vizinho e o outro, na interconexão direta de todos os nós.

Na grade, cada elemento da tabela possui a sua própria tabela de roteamento, contendo o nome do nó destino e uma métrica (distância em *hops* até o próximo elemento). No primeiro algoritmo, cada nó se conecta diretamente com o elemento vizinho. Assim, a rota para o vizinho torna-se rota padrão (*gateway*) para outros elementos da grade. Por exemplo, quando um elemento quiser requisitar um serviço, ele enviará uma requisição para o *gateway*, que será responsável por repassá-la para os demais nós.

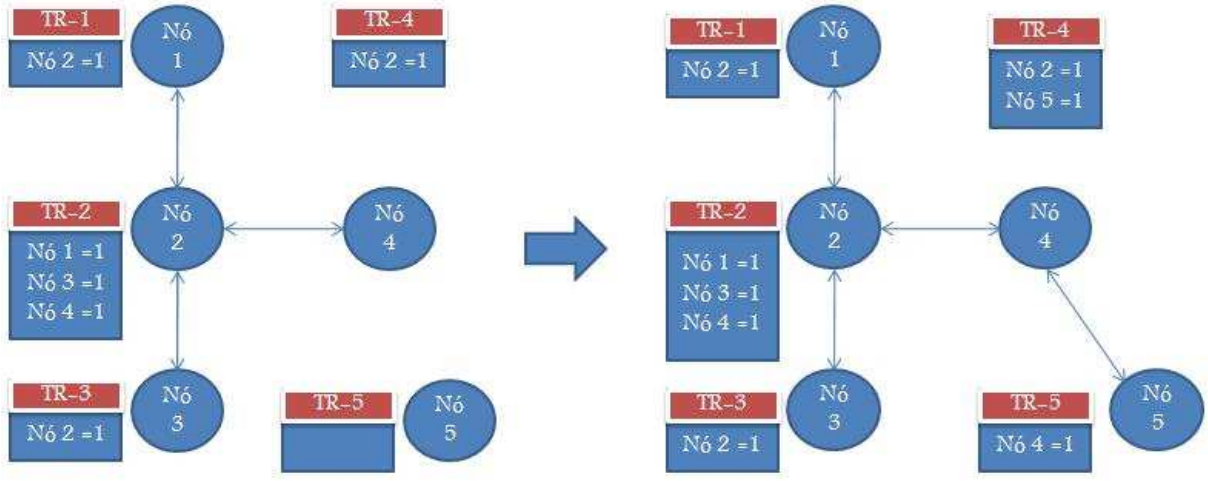


Figura 4.18: Algoritmo de roteamento baseado na interconexão direta com o nó vizinho

A figura 4.18 mostra o funcionamento deste algoritmo. Na parte esquerda da figura, o nó 5 está fora da grade. Os outros elementos da grade não possuem rota para o mesmo. Assim que o nó se insere na grade, mas precisamente através do nó 4, uma rota para o nó 5 é inserida no nó 4 e a deste no 5, ambas com métrica igual a 1, ou seja, diretamente conectada.

Há uma abordagem um pouco diferente no outro algoritmo. À medida que um elemento se insere na grade, todos os outros elementos adicionam uma rota direta (métrica igual a 1) para

ele. Isto fará com que a grade seja vista como um grafo completo. A difusão da informação da entrada ou saída de um elemento na grade é coordenada por este algoritmo de forma autonômica. Quando a difusão chegar ao fim e todos os nós forem informados das mudanças na topologia, chega-se a convergência.

A figura 4.19 ilustra esta situação. Em um primeiro momento (parte esquerda), o nó 5 está fora da grade. Ressalta-se que todos os outros elementos estão ligados diretamente (métrica 1). Em um segundo instante, há a adição deste elemento. Não interessa saber através de qual nó ele se conectou, pois as distâncias entre todos os elementos é a mesma. O primeiro elemento que perceber o pedido de inserção do nó 5, irá adicionar uma rota direta para ele, passará sua tabela de roteamento atual para o mesmo e por fim, informará aos outros integrantes da grade a existência do novo participante.

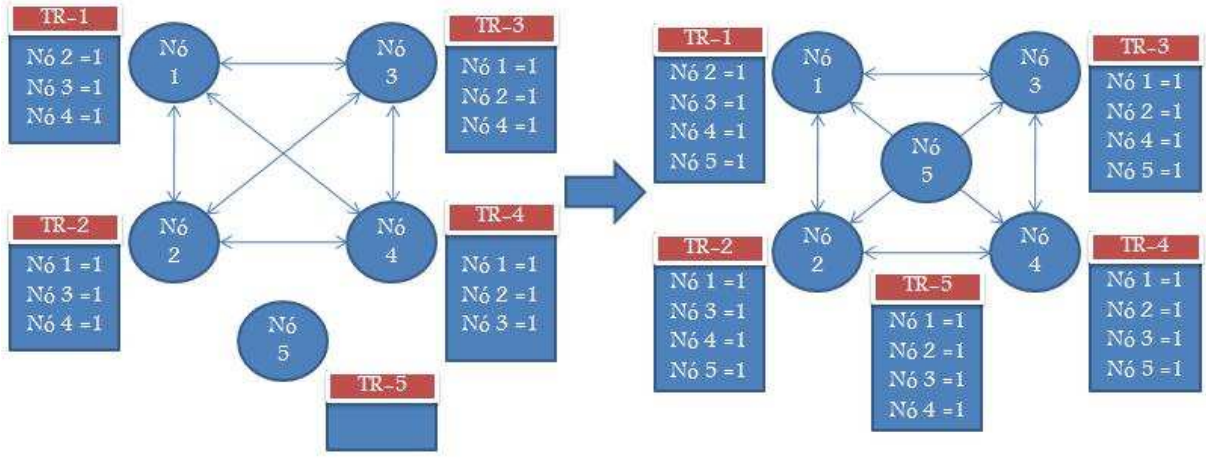


Figura 4.19: Algoritmo de roteamento baseado na conexão completa dos nós

A decisão sobre qual algoritmo utilizar na grade é de responsabilidade do criador da mesma, ficando implícito na criação do primeiro elemento. Ressalta-se que todos os nós da grade devem adotar o mesmo algoritmo, não sendo possível, portanto, que os dois interajam dentro do mesmo ambiente.

Cada algoritmo tem sua característica e pode se adequar melhor a uma situação específica. Por exemplo, caso a grade seja composta por muitos nós e a comunicação seja feita basicamente entre elementos adjacentes sem que haja necessidade de procurar serviços na grade com freqüência, a melhor solução seria a primeira. Porém, se a busca por serviços for grande e se desejar qualidade de serviço, como tempo de resposta, a melhor solução seria a segunda. Esta é exatamente a grande vantagem da abordagem do segundo algoritmo, no qual cada elemento pode se comunicar diretamente com qualquer outro na grade, tornando assim a busca e execução de serviços mais simples e rápida (tudo isso com um pequeno tempo de convergência). Isto

permite a esta abordagem proporcionar certas garantias de Qualidade de Serviço (QoS). Por este motivo, este algoritmo é o considerado padrão na instanciação dos elementos.

4.11 Estudo de Caso

Até este ponto, o trabalho discorreu sobre a teoria sob a qual este trabalho se baseou, detalhou a arquitetura, seus componentes e interações, além dos algoritmos de roteamento. Com o intuito de testá-la, decidiu-se implementá-la no Grid-M (ROLIM, 2007). Esta decisão baseou-se em algumas características oferecidas por este *middleware*: código aberto, facilidade para tratar com dispositivos de pequeno porte, disponibilidade de API amigável (FRANKE et al., 2007), portabilidade e outras mais, detalhadas na seção 3.8.

A arquitetura proposta em (ROLIM, 2007) não previa comportamento autonômico. Assim, esta (figura 3.3) teve de ser revista. A nova arquitetura do Grid-M, que suporta autonomia, é ilustrada na figura 4.20. Esta foi baseada na arquitetura discutida na seção 3.8.2, porém com algumas camadas remodeladas e outras criadas.

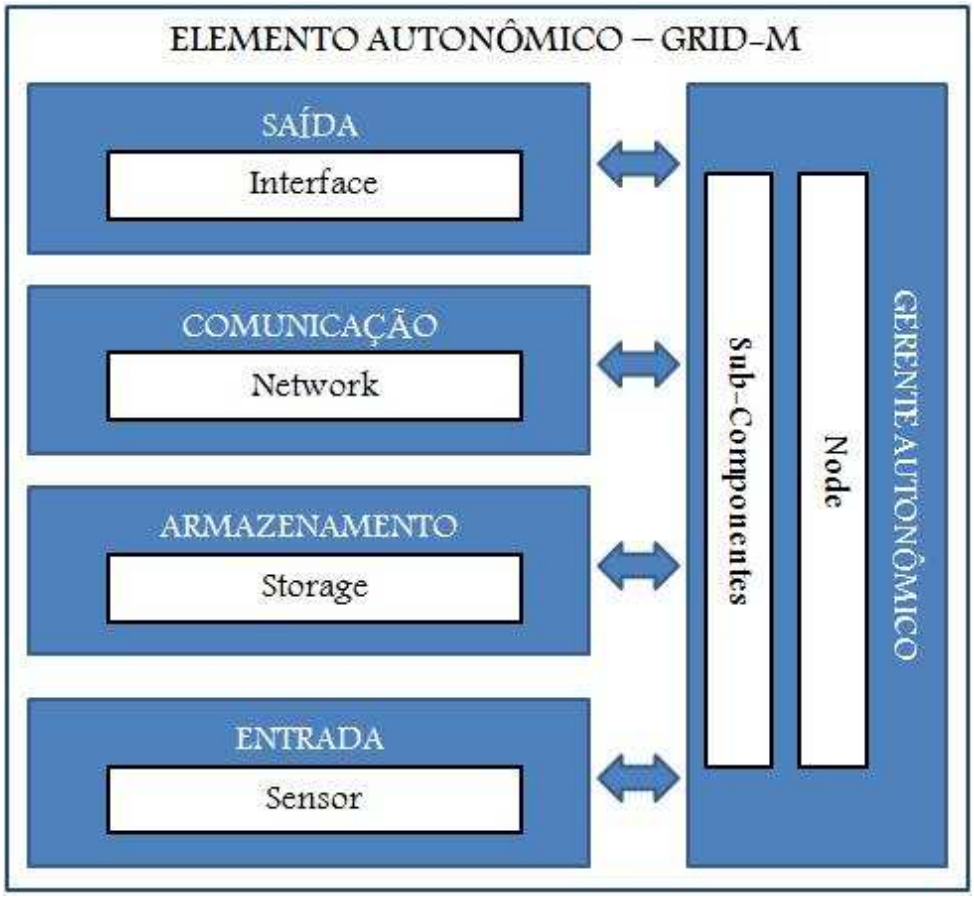


Figura 4.20: Nova arquitetura do Grid-M

Assim, as camadas desta nova arquitetura são:

- **Entrada**: é a camada mais baixa e que apresenta os sensores. Sua função é coletar os dados dos elementos gerenciados do nó e entregá-los ao gerente autonômico, que irá tratá-los;
- **Comunicação**: camada responsável pela comunicação entre os nós (roteamento e manipulação do protocolo de comunicação). Sua principal classe é a Network, que sintetiza as principais funcionalidades desta camada. A comunicação ocorre através do protocolo HTTP, assim como a antiga arquitetura;
- **Armazenamento**: camada responsável pelo armazenamento físico dos dados coletados e políticas de um elemento. Ressalta-se que esta camada trabalha em conjunto com o Gerente Autonômico, mais especificamente o componente de Gerência do Conhecimento;
- **Saída**: camada responsável por tratar a saída de dados, através da inferência e conversão dos dados no formato padrão da arquitetura. Sua principal classe é a Interface;
- **Gerente Autonômico**: esta camada oferece características de auto-gerenciamento ao sistema. Possui métodos para o gerenciamento interno de cada nó. Além disso, através da classe Node, fornece métodos relacionados com a criação, escalonamento, coordenação e execução de tarefas.

4.12 Testes quantitativos

Nesta seção, são apresentados resultados de alguns testes quantitativos obtidos em experiência de implementação que visam mostrar a eficiência do sistema proposto em diferentes situações de uso.

Com o intuito de testar o sistema proposto neste trabalho, foi criada uma grade composta com 30 nós. Este dispositivos são computadores pessoais com processador Intel Core Duo 1.66GHz, 2 GB de memória RAM e sistema operacional Windows XP. Ressalta-se que todos estão rodando os mesmo programas.

4.12.1 Tempo de convergência

Nesta seção, são feitos três testes, em tempos distintos, para cada um dos dois tipos de algoritmo. O intuito é testar os tempos de convergência. O tempo de convergência, para um

protocolo de roteamento, significa o tempo que ele gasta para que todas as tabelas de roteamento se atualizem quando da mudança de topologia (ex. inserção de um elemento na grade).

A princípio, o tempo de convergência poderia ser um gargalo, principalmente, no algoritmo no qual os nós estão todos diretamente conectados, já que as tabelas são propagadas por todos os elementos.

Analisando a figura 4.21, que mostra o tempo de convergência do algoritmo baseado na interconexão direta com o nó vizinho, percebe-se que o tempo de convergência é muito pequeno e, praticamente, constante (variação entre 10 e 14 ms). Isto ocorre porque o único processamento necessário é a inserção da rota do vizinho na tabela de roteamento. Nenhum dado referente à nova inserção é repassado adiante. O tempo foi coletado no momento da inserção de um novo elemento. Os elementos foram adicionados da seguinte forma: Nó2 se liga ao Nó1, Nó3 ao Nó2, Nó4 ao Nó3, e assim sucessivamente.

Figura 4.21: Tempo de convergência - Algoritmo baseado na conexão com o nó vizinho

Seguindo o algoritmo baseado na conexão completa dos nós como padrão, quando ocorrem mudanças de topologia, informações sobre as tabelas de rotas devem ser trocadas entre os elementos da grade. Por exemplo, quando um elemento se insere na grade, esta informação deve ser propagada a todos os elementos, não apenas ficar restrita àquele em que se conectou.

Os tempos de convergência deste algoritmo são ilustrados na figura 4.22. Os dados foram obtidos no momento da inserção do novo elemento (ex. o dado correspondente a grade formada por seis nós foi capturado no momento em que já havia uma grade composta por cinco elementos e um sexto foi inserido).

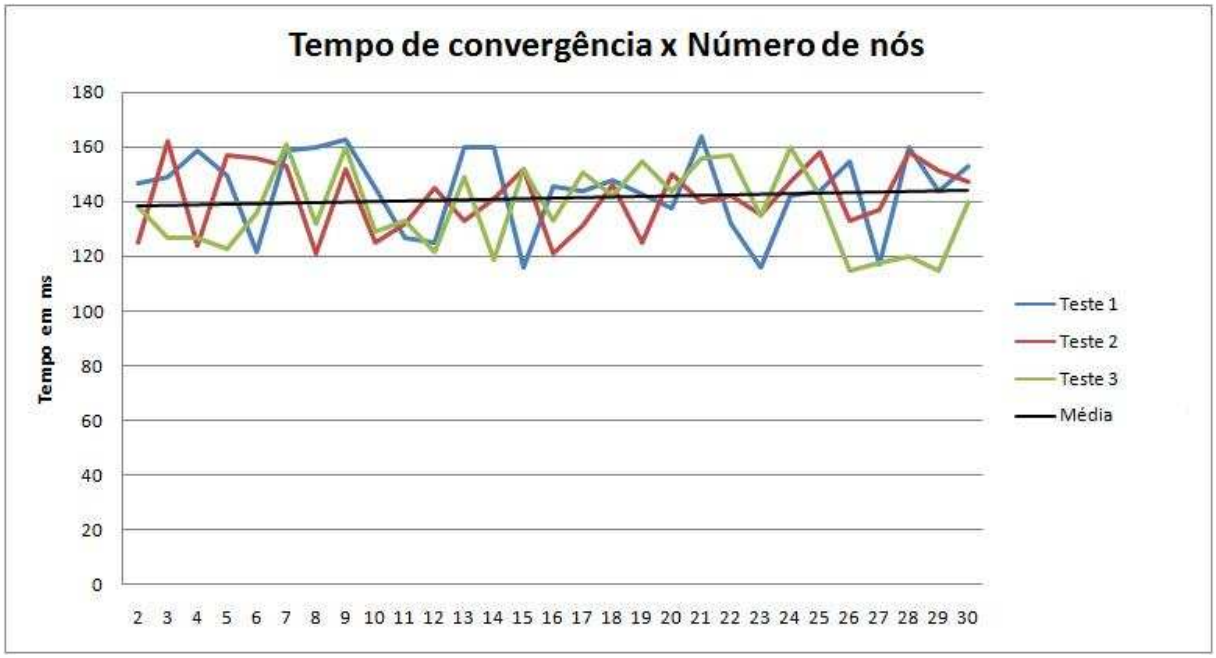


Figura 4.22: Tempo de convergência - Algoritmo baseado na conexão completa dos nós

A figura 4.22 mostra que o menor tempo de convergência após a inserção de um elemento foi no teste 2, após a inserção do nó 6, e o maior foi no teste 2 também, mas após a inserção do nó 10. Olhando a reta de tendência linear usada para traçar a média das convergências dos três testes, vê-se que à medida que são adicionados novos elementos, o tempo de convergência cresce, mas de forma muito pouco acentuada (média de 138 ms no início e 144ms no final). Este pequeno tempo gasto com convergência faz com que este algoritmo seja escolhido como padrão para o sistema proposto, já que os ganhos em tempo de resposta na execução de tarefas são grandes.

4.12.2 Tempo de resposta à execução de tarefas

Outro fator importante é o tempo de resposta a uma requisição de serviço. O tempo de resposta corresponde a soma de duas tarefas distintas: tempo de procura pelo serviço e tempo de execução.

O tempo de procura pelo serviço é o tempo gasto para que um requisitante realize uma busca na grade com o intuito de saber quem possui determinado serviço e qual possui o melhor percentual de recurso disponível (detalhado na seção 4.5.1). Assim, ao final da busca, tem-se o melhor candidato para a execução e uma tarefa é criada com ele como destino. Todo o processo de busca e redirecionamento das requisições é gerenciado e controlado pelo Gerente Autonômico, através dos sub-componentes detalhados nas seções 4.4 e 4.5.

Com a finalidade de testar os tempos de resposta às requisições de execução de serviços, utilizou-se a mesma estrutura do teste anterior (30 nós iguais), ilustrada pela figura 4.23. O teste consiste em o nó 1 requisitar o serviço à grade. O único nó que possui o serviço é o 30.

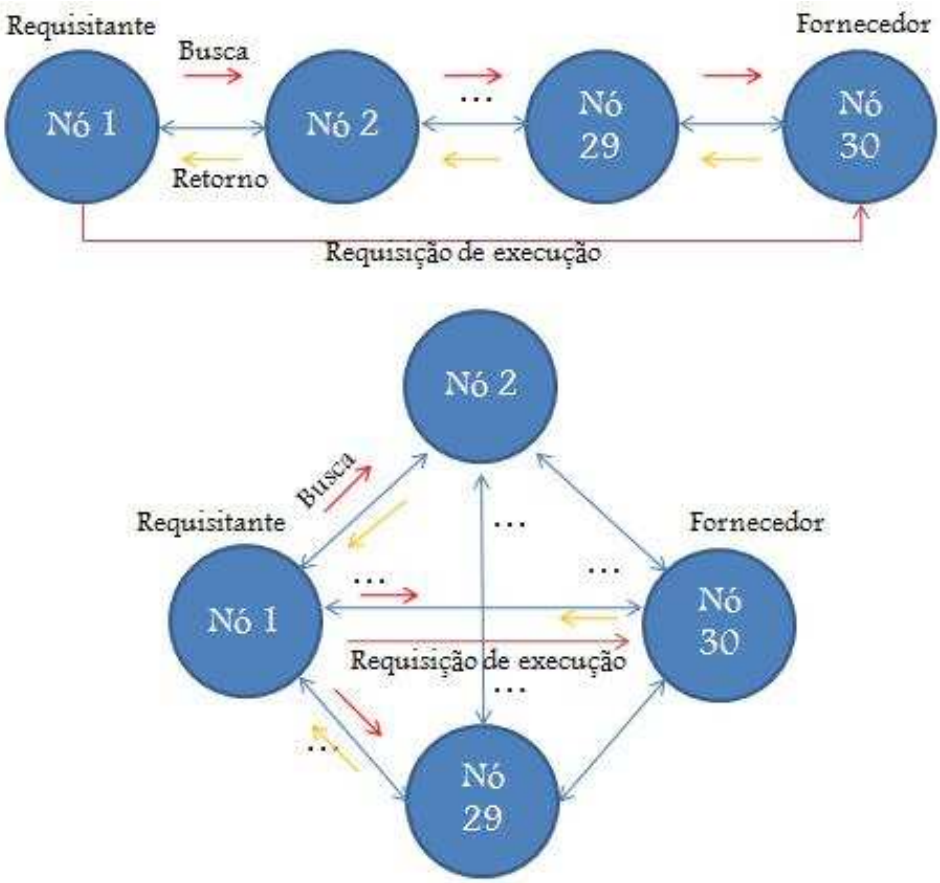


Figura 4.23: Topologias lógicas do teste de tempo de resposta

Deve ficar claro que este tempo de resposta depende do tipo de algoritmo de roteamento utilizado. Caso seja utilizado o algoritmo baseado na interconexão direta com o nó vizinho, a procura será mais lenta, se comparada com o algoritmo baseado na conexão completa dos nós. Isto é explicado porque o segundo possui uma visão completa da topologia (figura 4.23 parte inferior). Assim, a busca em todos os nós pode ser feita em paralelo (usando *threads*). Já utilizando o primeiro, a requisição de busca deve passar pelos nós intermediários, antes de chegar ao destino (figura 4.23 parte superior).

Após a busca, a tarefa é enviada diretamente para o elemento com melhor condição de executar o serviço, já que o endereço do destino é conhecido. Assim, o tempo de envio e de execução da tarefa não é influenciado pelo tipo de algoritmo de roteamento.

Os resultados alcançados, utilizando os dois algoritmos de roteamento, estão expostos na figura 4.24.

Figura 4.24: Resultado dos tempos de resposta

Como esperado, o tempo de resposta, utilizando o algoritmo baseado na interconexão restrita ao nó vizinho, é maior que o da outra abordagem. Como explicado anteriormente, isto é ocasionado pelo maior tempo de busca do primeiro algoritmo.

4.12.3 Eficácia da replicação de serviços

No teste anterior, fez-se com que o elemento 1 requisitasse um serviço à grade. Após a busca, constatou-se que somente o nó 30 oferecia determinado serviço. Como era um teste e sabia-se que a priori havia somente um solicitante (o nó 1), descarta-se a hipótese do nó 30 ser um gargalo por estar sobrecarregado executando o serviço oferecido. Porém, o que ocorreria se os outros 29 nós solicitassem o mesmo serviço? Neste caso, teria-se uma grande possibilidade do elemento 30 não conseguir responder a todas as requisições da melhor maneira possível, prejudicando o desempenho de toda a grade. Neste momento, observando que está sobrecarregado, o nó 30 enviaria uma requisição de replicação com a finalidade de encontrar outro elemento disponível a oferecer este serviço.

Vale lembrar que a replicação só é necessária uma vez e, após isto, o nó que recebeu o serviço passará a atender solicitações como provedor do serviço também.

A figura 4.25 mostra, através de dados retirados dos *logs* do Grid-M, como ficou a utilização dos recursos destas máquinas ao longo do tempo em quatro dos trinta elementos da grade. Ressalta-se que o nível de recurso dos quatro elementos são iguais a 5 com N (limite de recursos

livre) igual a 90.

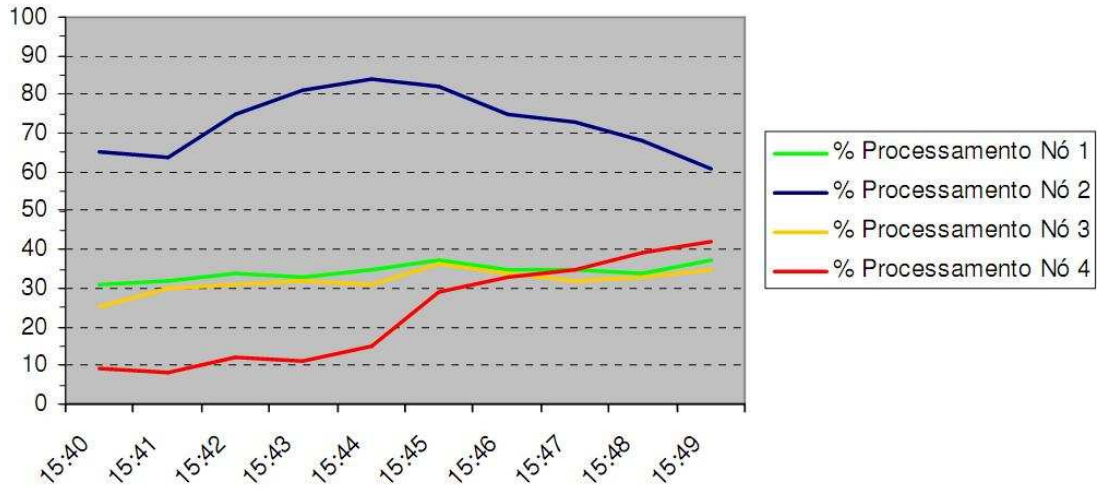


Figura 4.25: Utilização dos recursos dos nós e replicação de serviços

Neste teste, os nós 1 e 3 fazem requisições o tempo todo de um serviço que inicialmente somente o nó 2 possui. Em determinado momento, este nó fica sobrecarregado (percentual de recursos livre se torna menor que 18%) e então ocorre uma replicação de serviço do nó 2, que possui o serviço, para o nó 4, que era o nó com mais recursos disponíveis naquele instante.

Após isto, as requisições de serviço passaram a ser respondida pelo nó 4 também, distribuindo o processamento destas requisições.

Analisando o gráfico, percebe-se que o algoritmo eliminou a eminente saturação do nó 2 e a possível criação de um gargalo na rede.

4.12.4 Monitoramento dos recursos do nó

O monitoramento é fundamental para que o sistema possa se auto-gerenciar. É através dele que se detectam gargalos (sobrecargas de serviços). O gerente autonômico realiza esta função através do componente Monitoramento de Utilização do Hardware (MUH), que armazena as informações em *logs*. O MUH realiza esta função de tal forma que permite o gerenciamento de ambientes heterogêneos, formados por computadores pessoais, assistentes pessoais digitais, sensores, entre outros.

Utilizando os dados coletados, criou-se uma interface na qual o usuário do elemento pode acompanhar o uso de recursos do mesmo. Além disto, é possível ter informações descritivas sobre alguns periféricos como o processador e memória. Alguns exemplos da interface e de como estes dados são exibidos, são mostrados nas figuras 4.26, 4.27, 4.28, 4.29, respectiva-

Address Width:	32
Data Width:	32
Description:	x86Family6Model15Stepping6
L2 Cache Size:	2048
L2 Cache Speed:	Not Available
Manufacturer:	GenuineIntel
Maximum Clock Speed:	1663
Name:	Intel(R)Core(TM)2CPUT5500@1.66GHz
Current Clock Speed:	1663

Figura 4.26: Descrição do processador do elemento

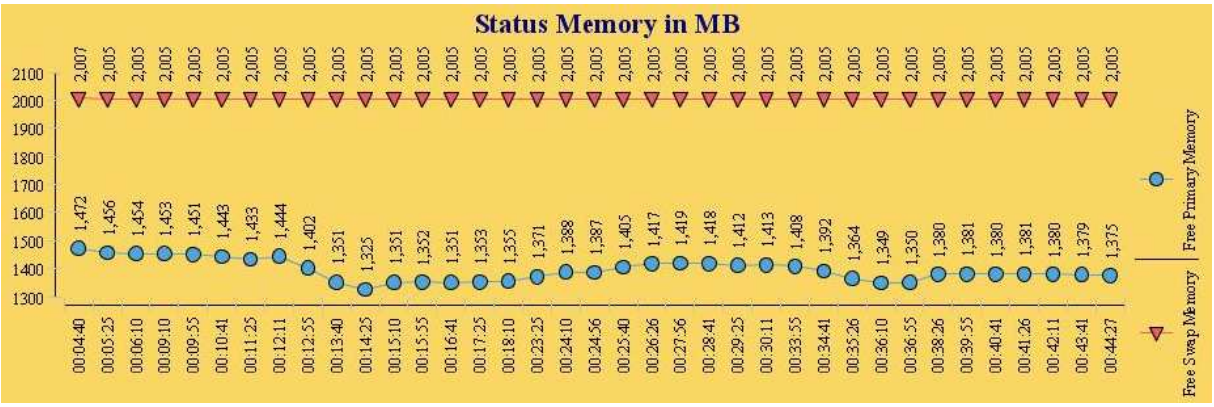


Figura 4.27: Quantidade de memória livre do elemento

mente, descrição do processador, quantidade de memória livre, quantidade de armazenamento disponível nos discos e quantidade de dados enviados e recebidos.

4.13 Considerações sobre o capítulo

No decorrer deste capítulo, a implementação e características específicas do sistema proposto neste trabalho foram detalhadas.

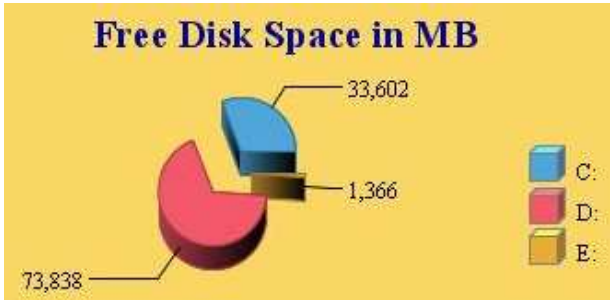


Figura 4.28: Quantidade de armazenamento disponível nos discos

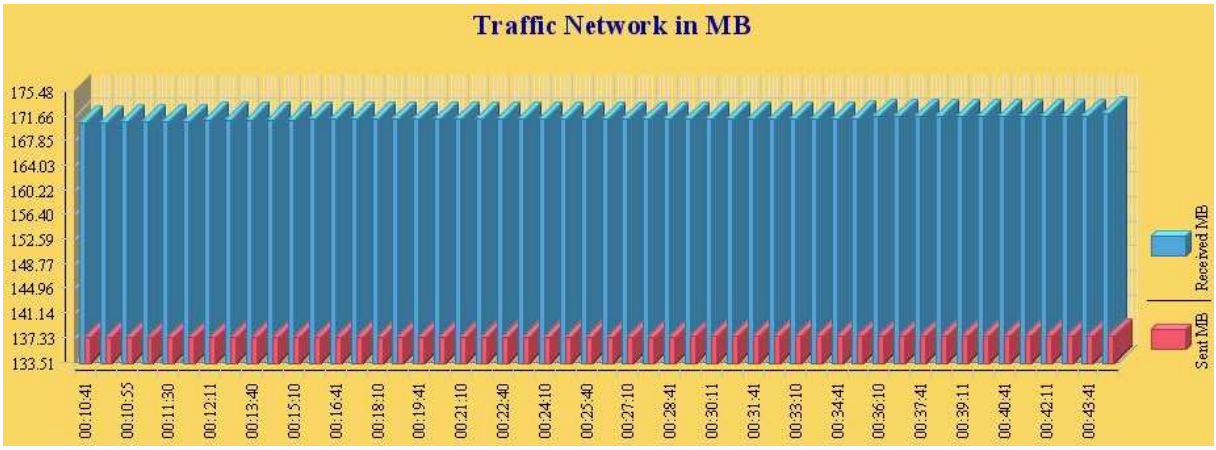


Figura 4.29: Quantidade de dados enviados e recebidos

Mostrou-se, através de um estudo feito nos trabalhos relacionados (seção 4.1), que trabalhos unindo as áreas de grade computacional e sistemas autonômicos ainda são poucos, mas que há inúmeras iniciativas, algumas com bastante sucesso, envolvendo os dois temas separadamente.

A composição do Elemento Autonômico, unidade funcional da arquitetura, foi definida neste capítulo. Este é formado por um Gerente Autonômico (responsável pelo auto-gerenciamento) e por elementos gerenciados, que agem no ambiente através de seus sensores e atuadores. Por sua vez, o Gerente Autonômico foi dividido em 7 componentes, facilitando a compreensão e modularizando as tarefas de gerenciamento. Estes componentes são responsáveis pela Gerência de Serviços, Redirecionamento das Requisições de Serviço, Replicação de Serviços, Monitoramento de Utilização do Hardware, Gerência da Tabela de Rotas, Tomada de Decisões e Gerência do Conhecimento. Devido à interação deles, o sistema oferece o comportamento autonômico desejado.

Definida a arquitetura, passou-se à implementação. Decidiu-se implementá-la no Grid-M, *middleware* que apresenta algumas facilidades interessantes para este trabalho. Assim, uma nova arquitetura foi definida para um nó no Grid-M, refletindo a necessidade de torná-lo, efetivamente, um elemento gerenciado. Detalhes desta nova arquitetura estão na seção 4.11.

Para mostrar a viabilidade do sistema, foi criado um ambiente de testes, no qual foram testadas três questões importantes para os sucesso do projeto: tempo de convergência (levando em consideração os dois algoritmos propostos), tempo de resposta à execução de tarefas e eficácia da replicação de serviços.

Os testes mostram que o sistema é viável, oferecendo gerência autonômica e garantias de qualidade de serviço, como a garantia que gargalos em serviços vitais serão evitados e o tempo de resposta será otimizado.

5 *Conclusão*

Neste trabalho, foi proposta um sistema para ambientes de grade computacional, baseada em uma base teórica sólida, com suporte à auto-gerenciamento. A grande questão a ser respondida era: *como fazer com que um ambiente bastante heterogêneo e de grande complexidade como as grades computacionais, não fique sujeito à gerência manual, que muitas vezes é ineficiente?* A solução proposta consiste na criação de elementos autonômicos, capazes de sentir o ambiente em que estão e atuar no mesmo de acordo com políticas pré-definidas. É exatamente a junção de vários elementos autonômicos que proporciona à grade, a capacidade de auto-gerenciamento desejada.

Para proporcionar auto-gerenciamento, o sistema deve suportar suas quatro sub-áreas: auto-configuração, auto-otimização, auto-regeneração e auto-proteção. Este trabalho abordou estas da seguinte forma:

- Auto-configuração: capacidade dada ao sistema de configurar-se sem nenhuma intervenção externa (como a humana). Esta característica é garantida pelo Gerente Autonômico. Exemplo disto é a capacidade do sistema de recriar ou replicar um serviço ao detectar uma falha (componente responsável pela Gerência de Serviços) ou um gargalo (componente Replicação de Serviços). Outra capacidade com esta característica é o algoritmo baseado na interconexão entre todos os nós. Ao ingressar na grade, um elemento conecta-se em outro nó. Este nó envia sua tabela para o recém chegado e informa aos outros integrantes da grade a existência deste novo elemento, que é adicionado na tabela. Tudo isto ocorre de forma automática;

- Auto-otimização: capacidade do sistema de otimizar-se de modo autonômico, permitindo sempre a melhor utilização possível dos recursos disponíveis. Exemplo desta capacidade é a replicação dos serviços sobrecarregados. Através dos dados coletados pelo componente responsável pelo monitoramento de utilização do hardware (MUH), o Gerente Autonômico detecta que um componente está sendo sobrecarregado por um serviço e decide replicá-lo em outro nó. Isto garante que um elemento não crie um gargalo na grade,

enquanto existirem outros nós ociosos;

- Auto-regeneração: capacidade do sistema de efetuar diagnósticos sobre si mesmo sem intervenção externa, com o intuito de encontrar erros ou subsistemas que não funcionam de modo correto. A Gerência da Tabela de Rotas é a componente do Gerente que apresenta esta característica. Ao checar que existem elementos remotos que não respondem às requisições, este componente se encarrega de eliminar a rota inválida na tabela, mantendo a integridade da mesma;

- Auto-proteção: capacidade do sistema de antecipar e encontrar eventuais intrusões ou violações de alguns parâmetros do sistema, permitindo um funcionamento seguro e controlado das suas funções. O sistema oferece suporte mas não implementa um serviço específico com este fim. A idéia é oferecer auto-proteção através do monitoramento dos recursos (componente MUH). (VIEIRA, 2007) apresenta uma proposta de detecção de intrusão em grades (GIDS) que aplica técnicas distintas de detecção de intrusão sobre os dados dos recursos colhidos do Grid-M. Com os dados, é criado um perfil de comportamento conhecido da aplicação e uma análise de conhecimento verifica possíveis quebras na política de segurança, analisando também por padrões de ataques conhecidos. Juntando essa proposta a este trabalho, atividade a ser feita futuramente, garante-se auto-proteção ao sistema.

Cobrindo as quatro áreas formadoras do auto-gerenciamento, consegue-se alcançar a autonomia desejada no início do trabalho.

Foram apresentados testes que comprovam a validade do sistema. Utilizando o algoritmo auto-configurável de roteamento, baseado na interconexão entre todos os elementos, com um pequeno tempo de convergência (144ms em uma grade com 30 nós), consegue-se ganhos grandes nos tempos de resposta (29ms de média, uma diferença de 979ms em uma grade com 30 nós se comparado com o outro algoritmo que não apresenta auto-configuração). Além disso, o sistema fornece uma interface, na qual consegue-se gerenciar os recursos do elemento.

Utilizar computação autonômica em ambientes de grade mostrou-se eficiente, pois além de permitir ao sistema se auto-recuperar de falhas, é possível garantir, através da auto-otimização, que não existirão gargalos em serviços muito requisitados.

5.1 Trabalhos Futuros

Este trabalho teve como escopo propor um sistema para gerência autonômica de grades computacionais e mostrar que a mesma é viável. Com a conclusão do mesmo, verificou-se a abertura de outras questões que podem ser tratadas em trabalhos futuros:

- criação de um serviço responsável pela segurança, baseado na proposta descrita em (VIEIRA, 2007);
- realização de testes com mais elementos na grade com a finalidade de testar a escalabilidade do sistema;
- execução de um estudo mais aprofundado sobre qualidade de serviço com o intuito de avaliar quais são os melhores parâmetros a serem controlados e priorizados;
- estabelecimento automático do nível do recurso. Atualmente, este nível é especificado na política do elemento;
- extensão da linguagem de especificação de políticas;
- questões relacionadas a segurança da infra-estrutura, principalmente, aquelas que visam proporcionar diferentes níveis de acesso, autenticação e autorização.

Referências Bibliográficas

ALLEN, G. et al. The gridlab grid application toolkit. *High-Performance Distributed Computing, International Symposium on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 411, 2002. ISSN 1082-8907.

ASSUNCAO, M. D. de. *Implementacão e Anáalise de uma Arquitetura de Grids de Agentes para a Gerência de Redes e Sistemas*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 03 2004.

BABAOGLU, O.; MELING, H.; MONTRESOR, A. Anthill: a framework for the development of agent-based peer-to-peer systems. *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, p. 15–22, 2002. ISSN 1063-6927.

BALEN, D. O.; FRANKE, H. A.; WESTPHALL, C. B. Gerenciamento de redes em grades computacionais: uma abordagem usando redes de sensores e contexto no monitoramento de redes. *XXVI Simpósio Brasileiro de Redes de Computadores. XIII Workshop de Gerência e Operação de Redes e Serviços (WGRS 2008)*, 05 2008.

BECKSTEIN, C. et al. Sogos - a distributed meta level architecture for the self-organizing grid of services. In: *MDM '06: Proceedings of the 7th International Conference on Mobile Data Management*. Washington, DC, USA: IEEE Computer Society, 2006. p. 82. ISBN 0-7695-2526-1.

BIGUS, J. P. et al. Able: A toolkit for building multiagent autonomic systems. *IBM Systems Journal*, v. 41, n. 3, p. 350–371, 2002.

BRASILEIRO, F. et al. Bridging the high performance computing gap: the ourgrid experience. In: *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid/First Latin American Grid Workshop (LAGrid07)*. Rio de Janeiro, Brazil: [s.n.], 2007. p. 817–822.

BRENNAND, C. et al. Automan: Gerência automática no ourgrid. In: *Anais do V Workshop de Grade Computacional e Aplicações*. Belém do Pará - BR: [s.n.], 2007.

BRENNER, W.; WITTIG, H.; ZARNEKOW, R. *Intelligent Software Agents: Foundations and Applications*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1998. ISBN 3540634118.

CHAO, K.-M.; JAMES, A. E. A framework for intelligent agents within effective concurrent design. In: SHEN, W. et al. (Ed.). *CSCWD*. [S.l.: s.n.], 2001. p. 338–343.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems (4th ed.): concepts and design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

DANTAS, M. *Computação distribuída de alto desempenho*. Rio de Janeiro, RJ, BR: Axcel Books do Brasil Editora, 2005.

DASHOFY, E. M.; HOEK, A. van der. Towards architecture-based self-healing systems. In: *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002. p. 21–26. ISBN 1-58113-609-9.

DAVIDSSON, P. Concept acquisition by autonomous agents: Cognitive modeling versus the engineering approach. In: *Lund University Cognitive Studies 12*. [S.l.: s.n.], 1992.

DIAO, Y. et al. Managing web server performance with autotune agents. *IBM Syst. J.*, IBM Corp., Riverton, NJ, USA, v. 42, n. 1, p. 136–149, 2003. ISSN 0018-8670.

ESTRIN, D. et al. Instrumenting the world with wireless sensor networks. *ICASSP*, IEEE Computer Society, Los Alamitos, CA, USA, v. 4, p. 2033–2036, 2001.

FAFNER. *Fafner-factoring via network-enabled recursion*. [S.l.]. Disponível em: <http://www.lehigh.edu/~bad0/fafner.html>.

FOSTER, I. *Internet Computing and the Emerging Grid. [S.l.]: Nature Web Matters*. [S.l.], 2000. Disponível em: <http://www.nature.com/nature/webmatters/grid/grid.html>.

FOSTER, I. What is the grid? a three point checklist. *GRIDToday*, 07 2002.

FOSTER, I.; IAMNITCHI, A. On death, taxes, and the convergence of peer-to-peer and grid computing. In: *International Workshop on Peer-to-Peer Systems (IPTPS'03)*. Berkeley, CA: 2nd Intenational Workshop on Peer-to-Peer Systems, 2003.

FOSTER, I.; KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. *Internacional Journal of Supercomputer Applications*, v. 11, n. 2, p. 115–128, 1997.

FOSTER, I.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. *Internacional Journal of Supercomputer Applications*, v. 15, n. 3, 2001.

FRANKE, H. A. et al. Grid-m: Middleware to integrate mobile devices, sensors and grid computing. *The Third International Conference on Wireless and Mobile Comunications - ICWMC*, 03 2007.

FRANKE, H. A. et al. Mobilidade em ambiente de grades computacionais. *XXIV Simpósio Brasileiro de Redes de Computadores. IV Workshop on Computational Grids and Applications - WCGA*, 06 2006.

GANEK, A. G.; CORBI, T. A. The dawning of the autonomic computing era. *IBM Syst. J.*, IBM Corp., Riverton, NJ, USA, v. 42, n. 1, p. 5–18, 2003. ISSN 0018-8670.

GARLAN, D.; SCHMERL, B. Exploiting architectural design knowledge to support self-repairing systems. In: *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*. New York, NY, USA: ACM, 2002. p. 241–248. ISBN 1-58113-556-4.

GEORGIADIS, I.; MAGEE, J. Self-organising software architectures for distributed systems. In: *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002. p. 33–38. ISBN 1-58113-609-9.

GLOBUS-OGSA. Towards open grid services architecture. Disponível em: <http://www.globus.org/ogsa/>.

GONZáLEZ-CASTA n. F. J. et al. Condor grid computing from mobile handheld devices. *SIGMOBILE Mob. Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 6, n. 2, p. 18–27, 2002. ISSN 1559-1662.

GRIDBUS. *Grid Computing and Distributed Systems Laboratory*. [S.l.], 2005. Disponível em: <http://www.gridbus.org/>.

GRIDFORUM. *Understanding Grids*. [S.l.], 2006. Disponível em: <http://www.gridforum.org/UnderstandingGrids% -/ggf_grid_understand.php>.

GUPTA, S.; JOSHI, A.; FININ, T. A Framework for Secure Knowledge Management in Pervasive Computing. In: *Proceedings of the Workshop on Secure Knowledge Management*. [S.l.: s.n.], 2008.

HARIRI, S. *AUTONOMIA : An Autonomic Computing Environment*. [S.l.], 2007. Disponível em: <http://www.ece.arizona.edu/~hpdc/projects/AUTONOMI>.

HARIRI, S. et al. The autonomic computing paradigm. *Cluster Computing*, Kluwer Academic Publishers, Hingham, MA, USA, v. 9, n. 1, p. 5–17, 2006. ISSN 1386-7857.

HERRMANN, K.; MüHL, G.; GEIHS, K. Self-management: The solution to complexity or just another problem? *IEEE Distributed Systems Online (DSOnline)*, v. 6, n. 1, 2005. Disponível em: <http://www.vs.uni-kassel.de/publications/2005/HMG05>.

HINCHEY, M. G.; STERRITT, R. Self-managing software. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 39, n. 2, p. 107, 2006. ISSN 0018-9162.

HORN, P. *Autonomic Computing: IBM's Perspective on the State of Information Technology*. Armonk, NY, USA, 2001.

IBM. *IBM Grid computing*. [S.l.], 2001. Disponível em: <http://www-1.ibm.com/grid/>.

IBM. *An Architectural Blueprint for Autonomic Computing*. [S.l.], 2005. Disponível em: <http://www.ibm.com/developerworks/autonomic/library/ac-summary/ac-blue.html>.

IBM. *The Océano project*. [S.l.], 2005. Disponível em: <http://www.research.ibm.com/oceanoproject% -/>.

JOSEPH, J.; ERNEST, M. Evolution of grid computing architecture and grid adoption models. *IBM Systems Journal*, v. 43, n. 4, 06 2004.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 36, n. 1, p. 41–50, 2003. ISSN 0018-9162.

KUBIATOWICZ, J. et al. Oceanstore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 35, n. 11, p. 190–201, November 2000. ISSN 0362-1340.

KUMAR, S.; COHEN, P. R. Towards a fault-tolerant multi-agent system architecture. In: *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*. New York, NY, USA: ACM, 2000. p. 459–466. ISBN 1-58113-230-1.

LEGION. *World Wide Virtual Computer*. [S.l.], 2006. Disponível em: <http://legion.virginia.edu/>.

LEMOS, R. de; FIADEIRO, J. L. An architectural support for self-adaptive software for treating faults. In: *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002. p. 39–42. ISBN 1-58113-609-9.

LIM, H. B. et al. Sensor grid: Integration ofwireless sensor networks and the grid. In: *LCN '05: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*. Washington, DC, USA: IEEE Computer Society, 2005. p. 91–99. ISBN 0-7695-2421-4.

LIU, H. et al. An autonomic service architecture for self-managing grid applications. In: *GRID '05: Proceedings of the 6th IEEE/ACM Internation Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2005. p. 132–139.

LOHMAN, G. M.; LIGHTSTONE, S. S. Smart: making db2 (more) autonomic. In: *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*. [S.l.]: VLDB Endowment, 2002. p. 877–879.

LOPES, R. F.; SILVA, F. J. da Silva e; SOUSA, B. B. de. Mag: A mobile agent based computational grid platform. In: *Proceedings of the 4th International Conference on Grid and Cooperative Computing*. Beijing: Springer-Verlag, 2005. (Lecture Notes in Computer Science (LNCS Series)).

LRG. *Grid-m: Middleware for embedded and mobile grid computing*. [S.l.], 2006. Disponível em: <http://grid.lrg.ufsc.br>.

LUTHER, A. et al. *Alchemi: A.NET-based Grid Computing Framework and its Integration into Global Grids*. 2005.

MALATRAS, A.; HADJIANTONIS, A. M.; PAVLOU, G. Exploiting context-awareness for the autonomic management of mobile ad hoc networks. *J. Netw. Syst. Manage.*, Plenum Press, New York, NY, USA, v. 15, n. 1, p. 29–55, 2007. ISSN 1064-7570.

MALLADI, R.; AGRAWAL, D. P. Current and future applications of mobile and wireless networks. *Commun. ACM*, ACM, New York, NY, USA, v. 45, n. 10, p. 144–146, 2002. ISSN 0001-0782.

MCCANN, J. A.; HUEBSCHER, M. C. Evaluation issues in autonomic computing. *In Proceedings of Grid and Cooperative Computing Workshops (GCC)*, p. 597–608, 2004.

MICROSOFT. *Introduction to windows management instrumentation*. [S.l.], 2003. Disponível em: <http://www.microsoft.com/whdc/hwdev/driver/WMI/WMIintro.mspx>.

MICROSOFT. *Autoadmin*. [S.l.], 2005. Disponível em: <http//research.microsoft.com/dmx-/autoadmin/>.

MICROSYSTEMS, S. *Sun N1 Service Provisioning System*. [S.l.], 2008. Disponível em: <http://www.sun.com/software/products/service_provisioning/index.xml>.

NWANA, H. S.; HEATH, M. Software agents: An overview. *Knowledge Engineering Review*, p. 1–40, 1996.

OASIS. *eXtensible Access Control Markup Language (xacml) committee specification 2.0*. [S.l.], Fev 2005.

PATTERSON, D. et al. *Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies*. [S.l.], March 2002. Disponível em: <http://research.microsoft.com/~emrek/pubs/ROC_TR02-1175.pdf>.

ROLIM, C. O. *Uma arquitetura para submissão de aplicacões de dispositivos móveis e embarcados para uma configuracão de grade computacional*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 04 2007.

ROURE, D. D.; JENNIGS, N. R.; SHADBOLT, N. The semantic grid: A future e-science infrastructure. In: ____. [S.l.]: John Wiley and Sons, 2003. cap. Grid Computing: Making The Global Infrastructure a Reality, p. 437–470.

SKILLICORN, D. B. *Motivating Computational Grids*. [S.l.]: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002. ISBN 0-7695-1582-7.

STERRITT, R. Towards autonomic computing: Effective event management. *27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 40, 2002.

SYCARA, K. Multiagent systems. *AI Magazine*, v. 10, n. 2, p. 79–93, 1998.

TESAURO, G. et al. A multi-agent systems approach to autonomic computing. In: *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*. Washington, DC, USA: IEEE Computer Society, 2004. p. 464–471. ISBN 1-58113-864-4.

UNICORE. *UNIform Interface to COmputer Resources*. [S.l.], 2006. Disponível em: <http://www.unicore.org/>.

VALETTO, G.; KAISER, G. A case study in software adaptation. In: *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002. p. 73–78. ISBN 1-58113-609-9.

VIEIRA, K. M. *Uma proposta de aplicação paralela de técnicas distintas de detecção de intrusão em ambientes de grid*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 04 2007.

WEISER, M. The computer for the 21st century. *Scientific American*, February 1991. Disponível em: <http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html>.

WOLF, T. D.; HOLVOET, T. Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control. In: *First International Workshop on Autonomic Computing Principles and Architectures*. [S.l.: s.n.], 2003. p. 10.

WOOLDRIDGE, M. *Introduction to MultiAgent Systems*. [S.l.]: John Wiley & Sons, 2002. Paperback. ISBN 047149691X.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, v. 10, p. 115–152, 1995.

WYATT, E. *Beacon Monitor Operations on the Deep Space One Mission*. 1998.

YOU, Y.; CHOI, H.; HAN, D. Autonomic system management for the pda. *Consumer Electronics, 2008. ICCE 2008. Digest of Technical Papers. International Conference on*, p. 1–2, Jan. 2008.

Apêndice A – Dados dos tempos de convergência dos algoritmos de roteamento

Tabela A.1: Dados do algoritmo baseado na interconexão entre todos os nós (em ms)

Quant. Nós	**Teste1**	**Teste2**	**Teste3**
2	147	125	138
3	149	162	127
4	159	124	127
5	150	157	123
6	122	156	136
7	159	153	161
8	160	121	132
9	163	152	160
10	145	125	129
11	127	132	133
12	125	145	122
13	160	133	149
14	160	141	119
15	116	152	152
16	146	121	133
17	144	131	151
18	148	146	142
19	143	125	155
20	138	150	144
21	164	140	156
22	132	142	157
23	116	135	135
24	142	147	160
25	144	158	142
26	155	133	115
27	117	137	118
28	160	158	120
29	144	151	115
30	153	147	140

Tabela A.2: Dados do algoritmo baseado na conexão com nó vizinho (em ms)

Quant. Nós	**Teste1**	**Teste2**	**Teste3**
2	12	11	11
3	13	11	13
4	13	12	10
5	14	13	14
6	12	14	11
7	10	11	11
8	12	13	14
9	13	11	10
10	10	10	10
11	11	12	11
12	12	13	12
13	11	10	14
14	11	13	10
15	13	12	13
16	10	12	12
17	11	11	14
18	10	10	11
19	14	10	12
20	10	13	10
21	11	11	13
22	11	12	12
23	12	14	12
24	13	12	13
25	13	12	14
26	10	13	11
27	10	13	11
28	10	10	13
29	11	11	10
30	13	12	12

Apêndice B – Dados da utilização dos recursos e replicação dos serviços

Tabela B.1: Dados da utilização dos recursos e replicação dos serviços (em %)

	% Nó 1	**% Nó 2**	**% Nó 3**	**% Nó 4**
15:40	31	65	25	9
15:41	32	64	30	8
15:42	34	75	31	12
15:43	33	81	32	11
15:44	35	84	31	15
15:45	37	82	36	29
15:46	35	75	34	33
15:47	35	73	32	35
15:48	34	68	33	39
15:49	37	61	35	42