UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JOSIANE ORTOLAN COELHO

Elemento Autonômico para Processos de Monitoração Adaptativa de Redes

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Profa. Dr. Liane Margarida Rockenbach Tarouco
Orientador

Prof. Dr. Luciano Paschoal Gaspary
Co-orientador

Porto Alegre, dezembro de 2008

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Coelho, Josiane Ortolan

Elemento Autonômico para Processos de Monitoração Adaptativa de Redes / Josiane Ortolan Coelho. – Porto Alegre: PPGC da UFRGS, 2008.

66 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2008. Orientador: Liane Margarida Rockenbach Tarouco; Co-orientador: Luciano Paschoal Gaspary.

1. Computação Autonômica. 2. Gerência de Redes. 3. Aprendizado por Reforço. I. Tarouco, Liane Margarida Rockenbach. II. Gaspary, Luciano Paschoal. III. Título.

“Embora ninguém possa voltar atrás e fazer um novo começo,
qualquer um pode começar agora e fazer um novo fim. ”
— CHICO XAVIER

AGRADECIMENTOS

Ao meus familiares, em especial a minha Mãe Reni e meu Pai João Guilherme, por oferecer todas as condições necessárias para eu alcançar meus objetivos. Eles são minha verdadeira fonte de inspiração devido à sua força e à sua determinação na busca pelos seus ideais. Amo muito vocês!

Ao meu Co-Orientador Luciano Paschoal Gaspary por ter-me "adotado" e acreditado na possibilidade de fazer um bom trabalho. Agradeço-te pela orientação, pelas críticas (e quantas foram...), pelas conversas, pelas sugestões que com certeza me tornaram uma profissional melhor a luz da sua competência. A minha Orientadora Liane M. R. Tarouco por ter me aceito no mestrado e permitir a realização desse trabalho em conjunto com o professor Luciano. Meu agradecimento especial ao colega Bruno Castro da Silva por todas as conversas pelas madrugadas geladas sobre Aprendizado por Reforço. Mesmo fazendo o doutorado nos EUA teve tempo e, principalmente, paciência em responder todas as minhas dúvidas.

Aos meus colegas do tempo da graduação e agora também de mestrado que me incentivaram a "entrar nessa" junto com eles. Em especial a turma do incentivo mútuo, das conversas sobre o andamento ou sobre o empacamento da dissertação, das ajudas com padrão latex, das simples conversas sobre a vida e o acaso: André Spritzer, Carolina Ming Chiao, Júlio Gerchmann, John Kliff, Priscilla Kurtz e Giovane Moura.

Ao apoio concedido pela Digitel, pela T&T e pela HP pela disponibilidade de tempo e de alguns recursos para a realização do mestrado. Aos pesquisadores Mícheál Ó Foghlú, Elyes Lehtihet e Nazim Agoulmine por enviar gentilmente seus artigos, os quais foram usados neste trabalho. Por fim, a uma pessoa especial que no final do mestrado, na parte mais crítica diria, soube me fazer sorrir “em muitas tardes vazias e me fez valer os dias...”, gracias Eduardo Costa Lopes!

A todos, ***Muito Obrigado***!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BDF	Breadth First Search
CA	Computação Autonômica
CHOP	Configuration, Healing, Optimization and Protection
CP	Contador de polling
DTMF	Distributed Management Task Force
GAP	Generic Aggregation Protocol
HTTP	Hypertext Transfer Protocol
IA	Inteligência Artificial
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ITU	International Telecommunication Union
MIB	Management Information Base
OMG	Object Management Group
SNMP	Simple Network Management Protocol
STP	Spanning Tree Protocol
TMF	TeleManagement Forum
UDP	User Datagram Protocol
VM	Virtual Machine
VoIP	Voice over IP
VPN	Virtual Private Network
WSPE	Web Service Polling Engine

LISTA DE FIGURAS

RESUMO

Estudos recentes sobre padrões de gerenciamento em redes de produção apontam que apenas um pequeno e estático conjunto de dados de gerenciamento tende a ser utilizado. Eles também revelam que o fluxo de dados de gerenciamento é relativamente constante e que as operações em uso para a comunicação agente-gerente são reduzidas a alguns, as vezes obsoletos, conjuntos. Essa realidade demonstra uma expressiva falta de progresso nos processos de monitoração, levando em consideração o seu papel estratégico e o potencial, por exemplo, para antecipar e prevenir falhas, perdas de desempenho e problemas de segurança em redes, serviços e aplicações. Uma das razões para tal limitação recai no fato de que o operador, ainda considerado um elemento fundamental no loop de controle, já não suporta o rápido crescimento tanto do tamanho quanto da heterogeneidade de ambos os componentes de *software* e de *hardware*, os quais constituem os modernos sistemas de computação em rede. Essa forma de "administrador no *loop* de gerenciamento" certamente dificulta a realização de adaptações oportunas nos processos de monitoração. Para resolver este problema, esse trabalho apresenta um modelo para monitoração adaptativa de redes, serviços e aplicações inspirado na abordagem de aprendizado por reforço. O modelo é analisado por meio da implementação de um protótipo de um elemento autonômico, o qual baseia-se em valores históricos, muitas vezes inesperados, obtidos de objetos gerenciados. Por meio do raciocínio sobre essas informações, o elemento autonômico dinamicamente amplia ou restringe o conjunto de objetos gerenciados a ser monitorado.

Palavras-chave: Computação Autonômica, Gerência de Redes, Aprendizado por Reforço.

Autonomic Element for Adaptive Network Monitoring Process

ABSTRACT

Recent investigations of management patterns in production networks suggest that just a small and static set of management data tends to be used, the flow of management data is relatively constant, and the operations in use for manager-agent communication are reduced to a few, sometimes obsolete set. This reality demonstrates an impressive lack of progress of monitoring processes, taking into account their strategic role and potential, for example, to anticipate and prevent faults, performance bottlenecks, and security problems. One of the key reasons for such limitation relies on the fact that operators, who still are a fundamental element of the monitoring control loop, can no longer handle the rapidly increasing size and heterogeneity of both hardware and software components that comprise modern networked computing systems. This form of human-in-the-loop management certainly hampers timely adaptation of monitoring processes. To tackle this issue, this work presents a model, inspired by the reinforcement learning theory, for adaptive network, service and application monitoring. The model is analyzed through a prototypical implementation of an autonomic element, which, based on historical and even unexpected values retrieved for management objects, dynamically widens or restricts the set of management objects to be monitored.

Keywords: Autonomic Computing, Network Management, Reinforcement Learning.

1 INTRODUÇÃO

A monitoração de redes, de serviços e de aplicações é uma das tarefas mais tradicionais da disciplina de gerenciamento e operação. Ela foi consolidada com o advento do Protocolo Simples de Gerenciamento de Redes (SNMP) no final dos anos 80. Desde então, têm sido observados a evolução do protocolo para satisfazer novos requisitos (por exemplo, desempenho e segurança), o aumento notável da quantidade e da variedade das informações de gerenciamento disponíveis, bem como o advento de novos *frameworks* para gerenciamento e de padrões recentes baseados em *Web Services* (PRAS, 2004; BULLARD; VAMBENEPE, 2006a,b; ARORA, 2008).

Ao mesmo tempo em que os avanços anteriormente mencionados são claramente reconhecidos, estudos recentes a respeito de padrões de gerenciamento em redes de produção (SCHÖNWÄLDER, 2007) sugerem que apenas um conjunto pequeno e estático de informações de gerenciamento tende a ser utilizado (por exemplo, informações associadas ao trafego de uma interface). Eles também revelam que o fluxo de dados de gerenciamento é relativamente constante e que as operações em uso para a comunicação agente-gerente são reduzidas a alguns, as vezes obsoletos, conjuntos (por exemplo, operações SNMP GetRequest ao contrário da operação mais eficiente GetBulkRequest). Essa realidade demonstra uma expressiva falta de progresso nos processos de monitoração, levando em consideração o seu papel estratégico, o potencial para antecipar e prevenir falhas, perdas de desempenho e problemas de segurança em redes, serviços e aplicações (para mencionar apenas algumas áreas as quais podem se beneficiar de uma eficiente monitoração).

Uma das razões para tal limitação recai no fato de que o operador, ainda considerado um elemento fundamental no *loop* de controle, já não suporta o rápido crescimento tanto do tamanho quanto da heterogeneidade de ambos os componentes de *software* e de *hardware*, os quais constituem os modernos sistemas de computação em rede. Essa forma de "administrador no *loop* de gerenciamento" certamente dificulta a realização de adaptações oportunas nos processos de monitoração. Por exemplo, é muito difícil determinar dinamicamente a quantidade de informações de gerência necessária para gerenciar redes, serviços e aplicações ao longo do tempo.

Tecnologias de auto-gerenciamento, especialmente aquelas que podem lidar com a evolução contínua dos atuais sistemas de computação e dos processos de Tecnologia da Informação, representam nesse contexto uma oportunidade para auxiliar os sistemas de gerência controlarem a si mesmos de forma autônoma. Diversos trabalhos têm sido realizados na tentativa de prover adaptação ao processo de monitoração de informações da rede por meio do uso de abordagens distintas tais como protocolo adaptativo (PRIETO; STADLER, 2007), escalonamento dinâmico de processos de monitoração (MACHIDA; KAWATO; MAENO, 2007) e promotores e inibidores bio-inspirados (DRESSLER, 2006). Apesar de possuírem o mesmo escopo, esses estudos não exploram a adap-

tação do conjunto de objetos gerenciáveis de maneira dinâmica.

Este trabalho apresenta um modelo para monitoração adaptativa de redes, inspirado na abordagem de aprendizado por reforço, capaz de ampliar e restringir o conjunto de objetos gerenciáveis para permitir o acompanhamento e a observação mais precisa. Para tal, esse modelo propõe a organização dos objetos gerenciáveis em formato de árvore para a representação dos estados de monitoração. Utiliza-se de mecanismos os quais empregam aprendizagem e adaptação para percorrer essa árvore. Para provar conceito e viabilidade técnica desse modelo, desenvolveu-se o protótipo de um elemento autonômico que foi utilizado como base para um conjunto de experimentos a partir de um estudo de caso.

A organização deste trabalho é realizada da seguinte forma: o capítulo 2 introduz a fundamentação teórica referente aos principais conceitos de computação autonômica, bem como suas propriedades fundamentais. Ele também aborda o conceito de aprendizado por reforço. No capítulo 3 são apresentados os trabalhos mais relevantes relacionados à adaptação e ao aprendizado empregados em pesquisas realizadas com foco em monitoração e alocação de recursos. No capítulo 4 é exposto o modelo proposto para proporcionar monitoração adaptativa. O capítulo 5 aborda a implementação do modelo e o capítulo 6 apresenta a sua avaliação parcial, baseado em um estudo de caso. Além disso, nesse capítulo, são comparadas as principais técnicas usadas no momento. Por fim, o capítulo 7 traz as conclusões da dissertação e enumera possíveis trabalhos futuros.

2 FUNDAMENTOS TEÓRICOS

Esse capítulo introduz os principais conceitos de Computação Autonômica e de Aprendizado por Reforço, os quais são base para a compreensão desse trabalho. A seção 2.1 apresenta a Computação Autonômica e suas principais características conhecidas como *Self-CHOP*. Nessa seção também são apresentados o conceito de elemento autonômico, o qual é consenso para todas as arquiteturas propostas para sistemas autonômicos, a importância de um modelo de informação e um exemplo de sistemas autonômicos. Na seção 2.2 é explicado o aprendizado por reforço destacando seus elementos fundamentais e as técnicas mais relevantes aplicadas aos métodos livre de modelo. Para o leitor já familiarizado com os conceitos abordados nas respectivas seções, sugere-se avançar para o próximo capítulo.

2.1 Computação Autonômica

Segundo Kephart e Chess (KEPHART; CHESS, 2003), Computação Autonômica caracteriza-se por sistemas que possuem a capacidade de se auto-gerenciarem de acordo com um conjunto de objetivos expressos em alto nível pelos administradores de tais sistemas. O objetivo dessa abordagem é prover capacitação do sistema para que ele possa se adaptar automaticamente às condições do ambiente, sem que ocorra a percepção do usuário e minimizando a intervenção humana no sistema. Dessa forma, desoneram-se os administradores dos detalhes de operação e manutenção do sistema, garantindo-se alta-disponibilidade dos serviços aos usuários. Os sistemas autonômicos possuem quatro características principais freqüentemente referenciadas como *Self-Configuration*, *Self-Healing*, *Self-Optimization* e *Self-Protection* (*Self-CHOP*) (KEPHART; CHESS, 2003), definidas abaixo.

- **Auto-configuração:** corresponde à capacidade de adaptação e integração, ambas automáticas e dinâmicas de acordo com políticas expressas em alto nível de abstração. Tais políticas representam o comportamento desejável ou esperado do sistema. Essas características de auto-configuração são diretamente impactantes aos fatores de qualidade do sistema tais como manutenibilidade, funcionalidade e portabilidade.

- **Auto-reparo:** o sistema autônomo deve ser capaz de descobrir, diagnosticar e reparar os problemas resultantes tanto de *software* quanto de *hardware*. Logo, o objetivo principal é maximizar a disponibilidade, a gerenciabilidade e a confiabilidade do sistema, sem transparecer ao usuário a ocorrência de falhas.

- **Auto-otimização:** consiste na habilidade de maximizar, de maneira eficiente, a alocação e a utilização de recursos para satisfazer diferentes usuários. O sistema, continuamente, procura aumentar suas operações, identificando e capturando oportunidades de melhorar o seu desempenho. Um modelo bastante usado na alocação de recursos é baseado nas funções de utilidade.

- **Auto-proteção:** refere-se à capacidade de estabelecer confiança, antecipar, detectar e recuperar o sistema de problemas decorrentes de ataques maliciosos ou de seqüência de falhas remanescentes de ações incorretas de auto-reparo. Para isso, o sistema usufrui de alertas precoces para antecipação e, conseqüentemente, prevenção de falhas.

Além dessas características mencionadas anteriormente, tem-se outras quatro propriedades secundárias responsáveis por conferir pró-atividade ao sistema, introduzidas por Salehie e Tahvildari (SALEHIE; TAHVILDARI, 2005): auto-consciência, integração, consciência de contexto e antecipação. A auto-consciência refere-se à capacidade do sistema estar ciente do estado, do seu comportamento e também da possibilidade de cooperação com outros sistemas. A característica integração corresponde à habilidade de interoperabilidade entre diferentes sistemas e sua portabilidade entre diversas plataformas. Consciência de contexto faz alusão à capacidade de conhecer seu ambiente de execução, bem como reagir a mudanças ocorridas no mesmo. Por fim, a antecipação significa que o sistema poderá antecipar a otimização dos recursos necessários para execução de alguma tarefa.

Existem diversas arquiteturas para sistemas autonômicos desenvolvidas por diferentes grupos de pesquisa, porém ainda não se tem um padrão consolidado. Todas elas são constituídas de elementos autonômicos, porém encontram-se algumas variações em relação à forma estrutural que compõe esses elementos. O escopo da próxima subseção irá se manter na apresentação de elementos autonômicos, considerando as características comuns às diferentes arquiteturas que são de suma importância para a compreensão do elemento autonômico proposto adiante. Para obter maiores informações sobre as principais arquiteturas existentes, recomenda-se como referência os artigos de Kephart (KEPHART, 2005), de Strassner (STRASSNER; AGOULMINE; LEHTIHET, 2006) e de Agoulmine (AGOULMINE, 2006). Outras duas propostas de arquiteturas para sistemas autonômicos são apresetadas em DARPA (DARPA, 2008) e em ACCA (ACCA, 2008).

2.1.1 Elemento Autonômico

Os sistemas autonômicos são constituídos por elementos autonômicos, os quais podem ser vistos como blocos básicos na construção desse tipo de sistema. Esses elementos gerenciam seu próprio comportamento e interagem com outros componentes, ou entidades, de acordo com políticas expressas por administradores ou requisitadas por outros elementos autonômicos. O comportamento de auto-gerenciamento surgirá dessa interação mútua entre os componentes, os quais podem ser, por exemplo, um repositório, uma base de dados, um servidor de aplicação, um *middleware*, um balanceador de carga de tarefas, entre outros.

Conforme ilustrado na figura 2.1, o elemento autonômico é composto pelos seguintes módulos: dispositivo gerenciado e gerente autonômico. O dispositivo gerenciado consiste na representação de um recurso passível de gerenciamento tal como equipamento, aplicação ou serviço. Por sua vez, o gerente autonômico é responsável pelo controle de um ou mais dispositivos gerenciados. Ele é composto por sub-módulos, representados pelos

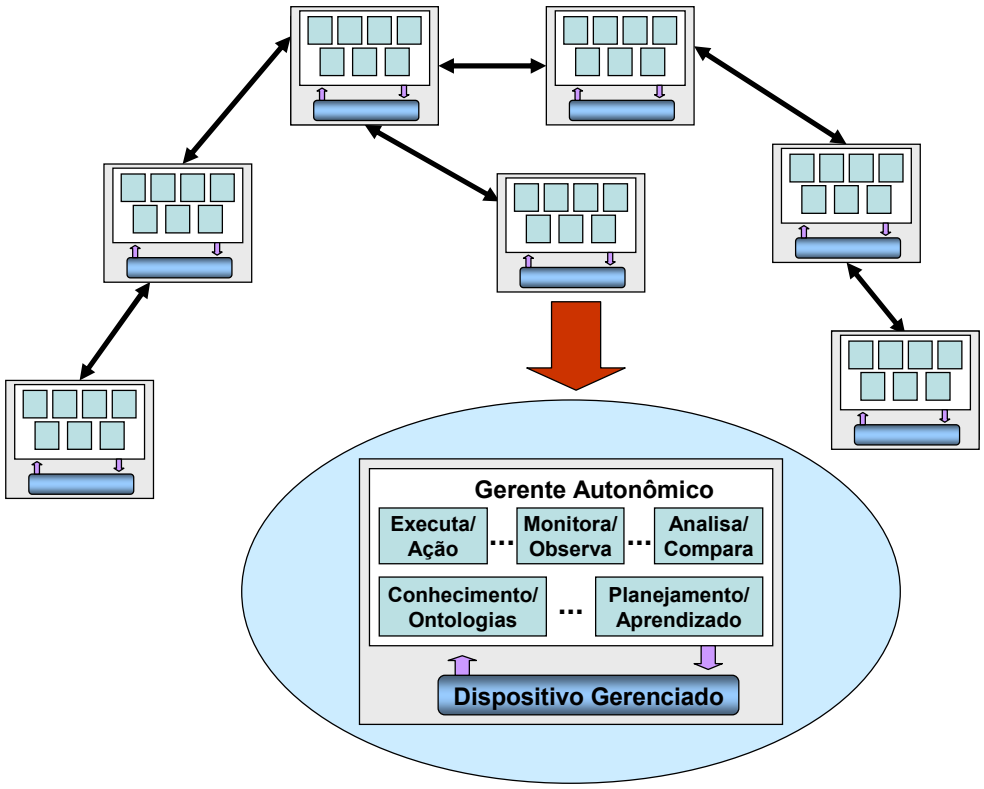


Figura 2.1: Representação estrutural dos elementos autonômicos e suas interações

quadrados na figura 2.1, cujas funcionalidades, analisadas em conjunto, possuem o objetivo de auto-gerenciar o elemento a partir de entradas provenientes da sua interação com o ambiente e com outros elementos. Esses sub-módulos possuem nomenclaturas distintas de acordo com as arquiteturas desenvolvidas pelos diferentes grupos de pesquisa. Um exemplo dessas nomenclaturas pode ser observado na figura 2.1 onde $Executa$ é adotada pelo grupo da IBM (KEPHART; CHESS, 2003), enquanto que *Ação* é utilizada pela arquitetura Focale (STRASSNER; AGOULMINE; LEHTIHET, 2006).

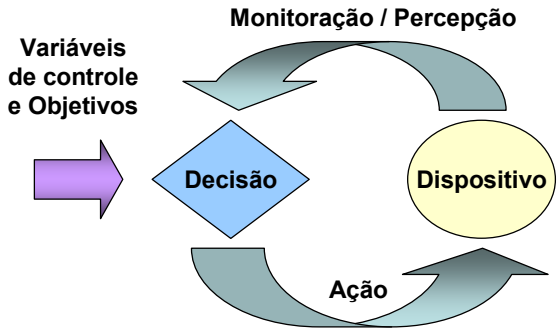


Figura 2.2: *Loop* de controle do elemento autonômico

Conforme ilustrado na figura 2.2, o *loop* de controle do elemento autonômico consiste nas interações entre os seus sub-módulos funcionais a partir da monitoração das informações originadas do dispositivo gerenciado bem como daquelas advindas do am-

biente. A partir de objetivos expressos e de variáveis de controle, essas informações são processadas/analisadas com a finalidade de serem utilizadas para tomar decisões no caso do elemento gerenciado não apresentar o comportamento esperado. Logo, identificado esse 'mau' comportamento como, por exemplo, uma falha no componente, o elemento deve disparar uma ação corretiva de modo a prover o estado de funcionamento desejado no ambiente. Nesse contexto de falha, a ação a ser tomada relaciona-se à capacidade de auto-reparo.

Como visto anteriormente, o elemento autonômico é responsável por gerenciar seu comportamento, bem como seu próprio estado de funcionamento interno. Além dessa gerência, existe ainda todo o controle das interações do elemento com ambiente e do elemento com outras entidades autonômicas. Por meio das interações entre elementos distintos, é possível obter o comportamento de cooperação o qual é necessário para a conquista de objetivos comuns ao sistema autonômico. Essas interações são baseadas em acordos pré-estabelecidos, os quais podem ser renegociados a fim de obter novas formas de colaboração entre os elementos autonômicos. No entanto, também pode existir o comportamento de autoridade de um elemento autonômico sobre outros através da delegação dos objetivos pela entidade mestre a fim de serem executados por uma ou mais entidades escravas. Observe a existência do conceito de hierarquia de diferentes níveis entre elementos autonômicos. Todavia, para serem factíveis esses vários tipos de interação entre os elementos autonômicos anteriormente expostos, necessita-se de um modelo de informações que considere os diferentes tipos de vocabulários existentes. Na próxima subseção, esse assunto será devidamente abordado.

2.1.2 Modelagem da Informação

Um dos desafios mais complexos na composição dos elementos autonômicos consiste na representação unificada do conhecimento. Essa necessidade advém da possibilidade de dispositivos com funcionalidades semelhantes, porém oriundos de fabricantes distintos, poderem, potencialmente, ser gerenciados pelo mesmo componente autonômico. Uma das alternativas possíveis consiste na simplificação e na unificação das tecnologias de informações entre os domínios de gerenciamento. No entanto, tem-se na prática a dificuldade das empresas em seguir padrões como, por exemplo, o caso das MIBs proprietárias que acabam sendo, muitas vezes, versões resumidas de MIBs padrões existentes, porém com semântica própria e interpretável apenas pelo fabricante.

Outra possibilidade provém da construção de vários domínios de conhecimento distintos, porém semanticamente equivalentes, a fim de proporcionar a interoperabilidade entre eles. Dessa forma, necessita-se de um modo dinâmico de integrar esses diferentes modelos para proporcionar uma base de conhecimento facilmente expansível. Atualmente, têm-se distintas especificações para o domínio de gerência providos pelas organizações TMF (TMF, 2008), DTMF (DTMF, 2008), OMG (OMG, 2008), IETF (IETF, 2008) e ITU (ITU, 2008). No entanto, ainda não se tem um modelo uniforme e definitivo de estruturação do conhecimento. Segundo Agoulmine (AGOULMINE, 2006), a importância da modelagem de informação pode ser mensurada em relação aos benefícios que ela pode propiciar, entre os quais destacam-se:

- *interoperabilidade entre dados*: consiste em manter o significado original dos dados através dos diferentes contextos de negócios, das estruturas de dados e dos tipos de esquemas;

- *interoperabilidade de processo*: permite expressar processos específicos em termos

de outros, por meio da inferência de significado a partir de modelos de processos e de meta-dados contextuais;

- *interoperabilidade de interfaces e serviços*: provê a busca, a associação e a comunicação com novos serviços sem a necessidade de codificação adicional;

- *interoperabilidade de aplicação*: proporciona acesso aos métodos, às transações e às chamadas à API entre aplicações distintas e independentes da plataforma utilizada;

A realidade atual da disciplina de gerência requer a integração conceitual dos diferentes padrões de gerenciamento na forma de uma base de conhecimento dinamicamente expansível. Com isso, torna-se possível alcançar a interoperabilidade entre domínios de conhecimento semanticamente equivalentes. A organização desses domínios de conhecimento por meio de um modelo uniforme objetiva assegurar a realização de consultas, em tempo de execução, aos objetos gerenciados sem implicar na adoção de um formato ou de um vocabulário em particular.

2.1.3 Exemplo de Sistema Autonômico

Com o objetivo de ilustrar alguns dos conceitos apresentados nas subseções anteriores, escolheu-se exemplificar os sistemas autonômicos por meio de sua modelagem, utilizando-se do ambiente de gerência de uma operadora de telefonia. Esse cenário da operadora de telefonia é muito rico para ser explorado, devido à grande diversidade tanto de serviços oferecidos quanto de equipamentos e tecnologias empregadas.

Analisando o cotidiano desse ambiente, os serviços prestados pela telefonia devem estar disponíveis aos usuários 24x7x365, devido às exigências dos contratos de prestação de serviços de alta-disponibilidade. Além disso, tem-se a penalização prevista pela legislação pelo tempo que o serviço de telefonia deixou de ser prestado. Imagine uma rede de equipamentos interconectados, nas quais estão presentes dispositivos responsáveis por concentrar uma grande quantidade de aplicações (VPN, voz e dados) de diferentes localidades e clientes, também conhecidos como nós de borda da rede. No caso de falha em algum nó desse tipo, devem-se encontrar rotas alternativas capazes de suportar essa demanda de serviços. No entanto, sabe-se que nem sempre os equipamentos estão configurados de acordo com as exigências necessárias para tornar apto o seu uso para determinado tipo de aplicação como, por exemplo, VPN.

Se existisse um sistema autonômico para a operadora de telefonia, o primeiro passo consistiria na identificação dessa situação de falha, referente à propriedade de consciência de contexto (seção 2.1), a fim de diagnosticar e tentar o reparo do equipamento através do reenvio de configurações padrões armazenadas. Observe que nessa situação, além da característica de auto-reparo pode-se utilizar a auto-configuração (seção 2.1). Contudo, caso seja uma falha não solucionável por esse procedimento como, por exemplo, a danificação de um *hardware*, o próprio sistema deve buscar rotas alternativas na rede as quais podem ser compostas por dispositivos de fabricantes distintos.

Nesse caso, o sistema deve ter a percepção das diferentes nomenclaturas que envolvem o processo de configuração desses equipamentos e, também, de uma possível interoperabilidade entre as várias plataformas de gerência presentes na operadora. Assim, fazem-se necessárias propriedades de auto-consciência e de integração (seção 2.1), que usufruem de um modelo de informações (seção 2.1.2) para prover a comunicação entre as diversas

plataformas de gerência. Esse modelo de informações auxilia na disseminação de conhecimento entre os componentes do sistema, na qual os objetivos expressos em alto nível (por exemplo, reconfigurar VPN para cliente A) são refinados para configurações de baixo nível.

Retornando a falha no nó de borda, o procedimento de redirecionamento de serviços para uma nova rota pode acarretar em uma quantidade de dispositivos usados fim a fim superior àqueles previamente envolvidos e, aumentando, assim, o atraso do serviço. Uma vez que o equipamento que apresentava a falha retorna ao estado de disponível, deve-se realizar uma reorganização das rotas a fim de obter o melhor desempenho, referente à auto-otimização (seção 2.1). Esse procedimento também poderia estar associado ao ingresso de cada novo dispositivo ao sistema. Esse pequeno estudo de caso ilustra o emprego das propriedades *Self-CHOP* no dia-a-dia das operadoras de telefonia. É evidente que nesse nível de descrição não se tem por objetivo entrar em detalhes de implementação devido à sua complexidade. Contudo, deseja-se ressaltar a importância desse novo paradigma da Computação Autonômica no desenvolvimento de sistemas.

Como visto anteriormente, a evolução dos sistemas de gerência está diretamente relacionada à capacidade desses sistemas controlarem a si mesmos de forma autônoma. Para isso, tecnologias de auto-gerenciamento, especialmente aquelas capazes de prover adaptação e aprendizagem, estão sendo continuamente desenvolvidas e, muitas delas, inspiram-se em abordagens provenientes da Inteligência Artificial como, por exemplo, o Aprendizado por Reforço. A próxima seção apresenta conceitos que fundamentam o funcionamento da técnica de aprendizagem por reforço.

2.2 Aprendizado por Reforço

Aprendizagem consiste em transformações adaptativas no sistema de modo a adquirir capacitação para realizar a mesma tarefa, ou tarefas similares, de maneira mais eficiente. Existem diversos tipos de aprendizado os quais podem envolver a supervisão de uma pessoa ou não. Dentre eles, destaca-se o aprendizado por reforço (SUTTON; BARTO, 1998), o qual se baseia na experiência das interações de forma a mapear estados a ações por tentativa e erro, maximizando o desempenho geral, também conhecido como valor recompensa. Por exemplo, imagine uma criança dada uma situação nunca antes vivenciada por ela. Essa situação, tal qual andar de patins, possui passos que podem ser interpretados como estados e tipos de movimentos realizados pela criança correspondentes a ações. Como a criança sabe andar sem patins, ela tenta aplicar o conhecimento prévio em busca de obter o mesmo êxito de se locomover sem esse equipamento. Assim, os movimentos que ela conseguir aplicar para andar terão mais credibilidade para uso futuro em outras situações análogas a essa. No entanto, as ações que não resultam em benefícios como, por exemplo, uma queda são armazenadas de maneira não tão positiva.

O aprendizado por reforço é definido em razão da caracterização de um problema de aprendizagem, o qual pode ser modelado como um Processo de Decisão de Markov (SUTTON; BARTO, 1998). O processo de Markov consiste em uma seqüência de estados finita, na qual o valor de um estado no instante futuro está sujeito à observação do estado atual e à ação realizada pelo agente, sem depender dos valores dos estados anteriores. Esse processo é definido formalmente por $<S,A,T,R>$, no qual têm-se: um conjunto finito de estados S, um conjunto finito de ações A, um modelo de transições de estados T e uma função de recompensa R. O modelo de transição (T:Sx$A \rightarrow \prod(S)$) consiste no mapeamento de um estado de S para uma ação de A de acordo com uma matriz de

probabilidade $\prod(S)$. Quando esse modelo de transições existe apenas de forma simulada ou quando ele não for conhecido, então utiliza-se o aprendizado por reforço. A função de recompensa R denota o reforço recebido pelo agente por escolher uma determinada ação em um dado estado.

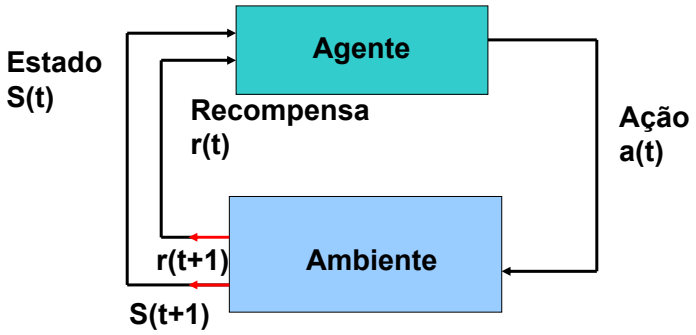


Figura 2.3: Interação Agente-Ambiente

A figura 2.3 ilustra o modelo padrão do aprendizado por reforço. Nesse modelo, um agente atua em um ambiente descrito por um conjunto de possíveis estados ($S(t)$) e pode executar, para cada estado, uma ação dentro de um conjunto de ações possíveis ($a(t)$), recebendo um valor de reforço ($r(t)$) a cada vez que executa uma ação. Este reforço indica o valor imediato da transição estado-ação-novo estado. Ao longo do tempo, este processo produz uma seqüência de pares estado-ação, e seus respectivos valores de reforço. O objetivo do agente é aprender uma política que maximize a soma esperada destes reforços em longo prazo. Esse processo tende a se repetir até que ocorra o final de um episódio de experimentação, ou até que o agente encontre um estado terminal, a partir do qual não é mais capaz de sair nem de receber reforços adicionais. Um episódio de experimentação consiste em uma subseqüência de interações agente-ambiente que termina de maneira natural.

2.2.1 Principais Elementos do Aprendizado por Reforço

Para compreender melhor a interação do agente com o ambiente, deve-se conhecer e entender o papel dos componentes formadores do aprendizado por reforço. Esses elementos são listados abaixo.

- Política π;
- Função recompensa $R(s,a)$;
- Função valor $V(s,a)$ ou $Q(s,a)$;
- Modelo do ambiente (opcional).

A política π é responsável por definir o comportamento do agente através de uma função que mapeia os estados para as ações correspondentes. Esse mapeamento estado-ação pode ser simplesmente uma tabela ou, em outros casos, pode envolver computações mais complexas como, por exemplo, funções polinomiais. As ações que constituem a política são alteráveis por meio da experiência de execução, garantindo ajustes frente às modificações ocorridas no ambiente. Uma política, para ser dita ótima (π*), deve apresentar

a melhor seqüência de estados e ações de forma a maximizar a soma dos reforços a serem obtidos. Por fim, a política pode ser estocástica, uma vez que pode-se associar uma probabilidade na escolha de uma ação a dado um estado s.

A função recompensa expressa o retorno do ambiente sobre o comportamento do agente quando executada uma ação, sendo inalterável pelo agente. Dessa forma, a recompensa indica os movimentos promissores imediatos. No entanto, o objetivo do agente é maximizar o ganho acumulado a longo prazo, descontando-se um fator γ, a fim de limitar a busca de solução para realizar uma tarefa. Logo, o fator γ pode ser visto como um fator de desconsideração das recompensas previstas nos passos futuros e indica-se seu uso para evitar a não convergência em modelos de tarefas descontínuas. Da mesma forma que a política, a recompensa também pode ser estocástica. A equação da recompensa (2.1) é expressa em função da soma dos reforços obtidos a partir do estado s, descontando-se γ que varia entre 0 e 1. Nesse caso, n representa o número de estados a ser considerado.

$$R_s = \sum_{k=0}^{n} \gamma^k r_{s+k} \tag{2.1}$$

A função valor indica o ganho total acumulado a partir de um estado, levando em conta os estados que sucedem o estado em consideração. Dessa forma, pode-se dizer que a estimativa do valor do estado não é independente, pois considera os valores dos estados futuros. A função valor (2.2) obedece a equação de *Bellman* que considera a utilidade de um estado como equivalente a soma do valor da recompensa imediata correspondente a este estado à utilidade descontada esperada dos próximos estados, dado que o agente continue realizando decisões a partir da mesma política. Estimar um valor de forma eficiente é a parte mais importante de quase todos os algoritmos de aprendizado por reforço. Observe que tem-se (2.3) definida em termos de $Q(s,a)$. Existe um método de aprendizagem denominado $Q\text{-}Learning$, apresentado na próxima subseção, que aprende a representação da ação-valor ao invés de aprender utilidades. Apesar disso, os valores de Q são diretamente relacionados com o valor de utilidade do estado, aplicando-se a equação (2.3) de forma equivalente à função valor.

$$V(s) = R(s) + \gamma \sum_{s'} T(s,a,s')V(s') \tag{2.2}$$

e

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s')Q(s',a') \tag{2.3}$$

Por fim, tem-se o modelo do ambiente como o único elemento opcional. Ele é responsável por imitar o comportamento do ambiente. O modelo pode ser completo ou pode ser incrementado no decorrer do processo de experimentação. A partir de um determinado estado e ação, o modelo completo é capaz de antecipar o próximo estado e a recompensa. Logo, esse componente pode ser empregado no planejamento de ações. Cabe ressaltar que, na abordagem de Aprendizado por Reforço, esse componente deve ser aprendido a

partir das interações do agente com o ambiente. A existência de um modelo único do ambiente é determinante para acelerar o aprendizado no caso de ambientes dinâmicos e estacionários. Contudo, no caso de ambientes não-estacionários, em geral, prefere-se ou métodos sem modelo ou métodos com modelos múltiplos, tantos quantos forem necessários para representar as diferentes dinâmicas que o ambiente pode assumir.

O aprendizado por reforço pode ser utilizado em três abordagens distintas: programação dinâmica, métodos Monte Carlo e métodos de diferença temporal. A programação dinâmica usufrui da existência de um modelo perfeito de ambiente para encontrar a utilidade ótima V^*. Já os métodos Monte Carlo não requerem um modelo e são conceitualmente simples. Apesar disso, eles não podem ser implementados de maneira a obter melhorias a toda interação, pois necessitam do episódio completo. Ou seja, deve-se realizar o processo de visitação dos estados até alcançar o estado terminal. Por fim, métodos de diferença temporal também não exigem a existência de modelos e são totalmente incrementais, uma vez que aproveita-se de estimativas parciais. Ou seja, são métodos mais adequados para ambientes dinâmicos e não-estacionários, pois tendem a aprender mais rapidamente. Na próxima subseção será apresentado um dos algoritmos mais importantes para os métodos de diferença temporal conhecido como $Q\text{-}Learning$.

2.2.2 *Q-Learning*

O algoritmo $Q\text{-}Learning$ (SUTTON; BARTO, 1998) baseia-se no aprendizado iterativo de uma política ótima π^* quando o modelo de ambiente não for conhecido. Ou seja, o algoritmo deve aprender uma função de avaliação ótima sobre todos os valores estado-ação que será posteriormente utilizada na escolha da melhor ação para uma dada situação. A função de estimação Q consiste na soma do valor estado-ação atual pela taxa de aprendizagem α que multiplica um termo, conforme pode ser verificado na equação (2.4). Esse termo é constituído pela soma da recompensa recebida pelo agente pela execução da ação a_t no estado s_t no instante t ao fator de desconto γ, multiplicado pela diferença entre o máximo valor estado-ação do estado sucessor e o valor estado-ação atual. A taxa de aprendizado α, normalmente definida entre 0 e 1, é responsável por amortizar o valor estado-ação aprendido.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_t + \gamma\left(max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right)\Big) \tag{2.4}$$

Observe que o valor estado ação Q é aprendido de forma gulosa e independe da política utilizada. Assim, o agente é capaz de aprender mesmo seguindo uma política aleatória, porém esse processo consome um tempo elevado para sua convergência, caso comparado a outros meios (ex: ϵ-Gulosa, *Boltzmann*, detalhados na seqüência).

O algoritmo 1 ilustra os principais passos da execução do Q-*Learning*. Inicialmente, determina-se o estado origem. Após, têm-se o processo da escolha da ação, seguida da sua execução e, posteriormente, analisa-se a recompensa recebida. Esses passos repetem-se até encontrar o estado terminal. Note que a atualização da função ação-valor obedece a equação anteriormente apresentada em (2.4). Uma consideração importante a ser realizada quando tem-se uma política é o conflito exploração *versus* aproveitamento. O agente deve lidar com o problema de existir um compromisso entre escolher a exploração de estados e ações desconhecidos, de modo a coletar nova informação, ou o aproveitamento dos valores dos estados e das ações que já foram aprendidos. Dessa forma, consideram-se

Algoritmo 1 *Q-Learning*

Seja: $Q(s, a)$ inicializado arbitrariamente, para todo s e para todo a

1: **Repita**
2: Defina um estado inicial s.
3: **Repita**
4: Escolha a a partir de uma política (Ex: ϵ-Gulosa ou *Boltzmann*).
5: Execute a e observe a recompensa r e o estado sucessor s_{t+1}.
6: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma\ maxQ(s_{t+1}, a_t) - Q(s_t, a_t))$
7: $s \leftarrow s_{t+1}$
8: **até** que s seja um estado terminal.
9: **até** fim Repita

as ações que irão promover altas recompensas de modo a maximizar os reforços acumulados. Sendo assim, por um lado o agente deve aprender quais ações maximizam os valores das recompensas obtidas no tempo e, por outro, deve agir de forma a atingir esta maximização explorando ações ainda não executadas ou regiões pouco visitadas do espaço de estados. Portanto, é importante estabelecer uma política mista de exploração e aproveitamento, que inclua na escolha certa probabilidade de se executar uma ação que não é considerada a melhor no estado atual, visando a aquisição de conhecimentos a respeito de estados ainda desconhecidos ou pouco visitados. Entre as políticas que mesclam essas duas abordagens mencionadas previamente destacam-se: ϵ-Gulosa e *Boltzmann*.

A política ϵ-Gulosa (SUTTON; BARTO, 1998) consiste na seleção de ações as quais já se conhece o alcance de boas recompensas durante certa quantia de tempo e, no tempo restante, busca-se aprender novas ações através da escolha aleatória. Desse modo, permite-se que ações não tão boas à primeira vista tenham a possibilidade de serem experimentadas e possam conduzir a ações com recompensas verdadeiramente ótimas. Existem diversas maneiras de implementar essa função de exploração (RUSSEL; NORVIG, 2005), sendo uma delas descrita abaixo:

$$f(u, n) = \begin{cases} \Re^+ & se\ n < N_e \\ u & \text{em caso contrário} \end{cases}$$

onde $\Re^+$ é a estimativa da melhor recompensa que pode ser obtida em qualquer estado, n consiste de um número aleatório entre 0 e 1, N_e corresponde a uma taxa de exploração que varia entre 0 e 1 e u representa a utilidade ótima aprendida previamente. Dessa forma, caso N_e seja igual a zero tem-se uma política totalmente gulosa. Porém, se N_e for igual a 1, existe uma alta probabilidade de exploração entre as ações possíveis para o estado analisado. Observe que, nessa abordagem, não existe diferenciação na escolha de ações as quais apresentam valores de recompensas muito ruins daquelas que aparentam ter valores de recompensa próximos ao ótimo. Ou seja, todas as ações, independente da estimativa de seus reforços, são escolhidas com a mesma probabilidade.

Uma outra abordagem alternativa ao método ϵ-Guloso consiste na seleção *softmax* (SUTTON; BARTO, 1998), na qual destaca-se o método que usa a distribuição de *Boltzmann*. Nessa distribuição, a probabilidade de escolha da próxima ação varia de acordo com o seu valor esperado de recompensa. Ou seja, existe a priorização para escolha das ações que apresentam valores mais elevados, em detrimento daquelas com valores muito baixos. A escolha da ação é realizada de acordo com a seguinte distribuição:

$$a = \frac{e^{\frac{Q_{t(a)}}{\tau}}}{\sum_{i=1}^{n} e^{\frac{Q_{i(b)}}{\tau}}}$$

onde n corresponde ao número de ações e τ é um parâmetro positivo de temperatura que pode decrescer com o tempo a fim de diminuir o grau de exploração. Quanto mais elevada for a temperatura, mais as ações tornam-se equiprováveis. Temperaturas mais baixas ressaltam as diferenças na probabilidade de seleção de ações. Quando τ tende a zero, a seleção *softmax* equivale à gulosa. Assim, esse método apresenta melhores resultados quando as ações que possuem maiores reforços são significativamente distantes daquelas que contém baixos valores. No entanto, quando os valores das ações são muito próximos, a convergência da solução tende a ser lenta.

2.3 Sumário

Este capítulo apresentou conceitos básicos sobre o paradigma de computação autonômica e sobre a abordagem de aprendizado por reforço.

Na subseção 2.1 foram introduzidas as principais características da computação autonômica e o componente básico para construção de sistemas autonômicos: o elemento autonômico. Além disso, foi salientada a importância da definição de um modelo unificado de informação. Esse modelo é um componente chave para regulamentar as interação entre os módulos funcionais internos ao elemento, bem como a interação entre outros elementos e, inclusive, com o ambiente. A partir dessas diferentes formas de interações, emerge o comportamento de autonomia, o qual é necessário para ter a auto-gestão no sistema autonômico. Por fim, apresentou-se, a título de ilustração, um exemplo de sistema autonômico relacionado à área de redes.

Em 2.2 foi apresentada a metodologia de aprendizado por reforço, utilizando-se do modelo de interação do agente com o ambiente para explicar a seqüência base de seu funcionamento. Também foram listados e detalhados os principais elementos do aprendizado por reforço, com foco nas suas responsabilidades. Finalizando, foi introduzido o método $Q - Learning$, o qual é indicado para resolução de problemas onde o modelo de ambiente não é conhecido. Além de aprender, esse método permite adaptar o conhecimento adquirido de acordo com a percepção de alterações ocorridas no ambiente.

3 TRABALHOS RELACIONADOS

Ainda é muito incipiente na literatura a aplicação da técnica de aprendizado por reforço para resolver problemas de gerência de sistemas. Em função disso, este capítulo descreve trabalhos que apresentam tentativas de adaptação do processo de monitoração de informações da rede por meio do uso de abordagens distintas. Após, apresentam-se dois trabalhos que empregam a técnica de aprendizado por reforço para prover alocação dinâmica de recursos e descoberta de menor caminho entre servidores vizinhos.

3.1 Protocolo Adaptativo para Monitoração Contínua da Rede com Objetivos Precisos

As soluções atuais de gerência para o processo de monitoração contínuo da rede recaem em um dilema fundamental: estimativa precisa das variáveis *versus overhead* de gerenciamento proveniente do tráfego e da carga de processamento dos dados. Na maior parte das soluções desenvolvidas, opta-se por proporcionar um controle qualitativo da precisão. Geralmente, esse controle é realizado através da agregação das variáveis gerenciadas e, posteriormente, do seu cálculo por meio de funções tais como soma, média, mínimo e máximo.

Segundo Gonzalez e Stadler (PRIETO; STADLER, 2007), a elaboração de um protocolo genérico de agregação com precisão controlável é uma alternativa para prover estimativas precisas sem acarretar necessariamente em *overhead* de tráfego de informações. O protocolo A-GAP, proposto pelos autores, baseia-se no *Generic Aggregation Protocol* - GAP, que é um protocolo distribuído e assíncrono para a construção e a manutenção de uma *spanning tree* do tipo BDF (*Breadth First Search*). A manutenção dessa árvore é realizada de modo similar ao algoritmo utilizado pelo *Spanning Tree Protocol* - STP.

Na figura 3.1, tem-se um exemplo de árvore de agregação das variáveis monitoradas. Cada dispositivo da rede é um nó responsável por executar um processo de gerência, o qual disponibiliza informações com propósito de monitoração. Assim, cada círculo na figura representa um processo de gerenciamento. Internos ao processo, existem um nó folha e um nó de agregação. Cada nó folha está associado a uma variável local, a qual representa um valor de estado ou um contador do dispositivo que é objeto de monitoração. O valor dessa variável é atualizado assincronamente. O nó de agregação mantém o valor do agregado parcial (AP_1) que corresponde a soma do valor do agregado parcial dos nós filhos ($AP_{1.1}$ e $AP_{1.2}$) em conjunto com a soma do valor da variável local (v_1), conforme ilustrado na figura 3.1. Atente ao fato de que essa topologia em forma de árvore independe da topologia física real da rede.

Cada nó da árvore também contém um filtro local que é responsável por permitir ou

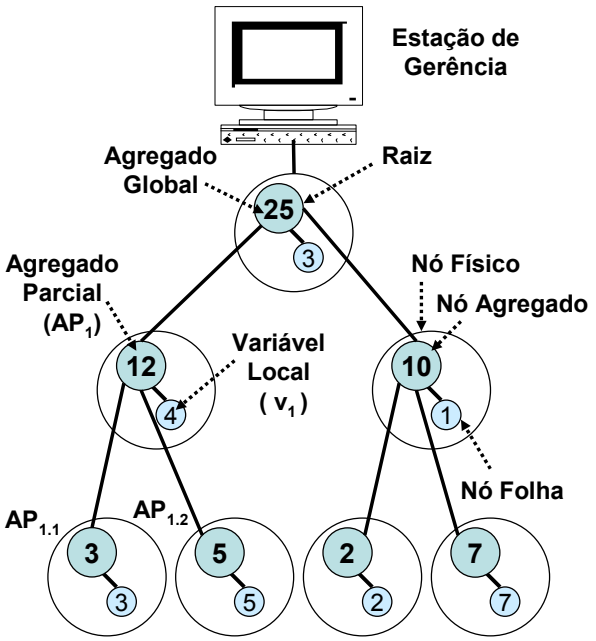


Figura 3.1: Árvore de agregação com função de agregação soma

não o envio de mensagens de atualização do novo valor da variável local (no caso de ser um nó folha) ou do agregado parcial para seu nó pai. A mensagem de atualização somente é enviada quando a diferença entre o valor anteriormente reportado e o valor atual excede o filtro local como mostra a figura 3.2. A linha tracejada corresponde à última atualização do valor, enquanto que a linha completa representa o valor do agregado parcial (ou do valor local da variável). A linha pontilhada, por sua vez, indica os limites do filtro. Observe que no instante ilustrado pela linha vertical tem-se o envio da mensagem contendo o novo valor e desloca-se o limite do filtro local. Esse deslocamento é realizado por meio do cálculo dinâmico da estimativa de erro.

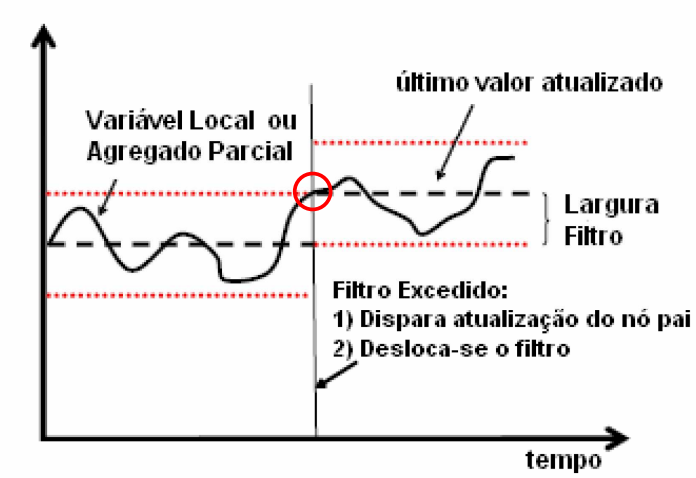


Figura 3.2: Filtro local ultrapassado no instante ilustrado pela elipse.

No experimento realizado, analisou-se a variável local correspondente ao número de fluxos HTTP que acessam um nó (na ordem de 20.000 fluxos por nó) no período de 200 segundos. O *delay* de comunicação, que consiste no tempo necessário para o envio de uma mensagem de atualização, é de 4 ms e gasta-se 1 ms no processamento de cada mensagem. Independente da topologia ser *grid* ou não e considerando um erro absoluto de 20 fluxos, constatou-se a redução de até 85% das mensagens de atualização comparando-se ao resultado obtido no experimento com erro absoluto igual 0 (que corresponde ao nó local enviar mensagem de atualização a cada alteração do valor da variável local). Em síntese, a adaptação alcançada por essa abordagem deu-se por meio da seleção das mensagens de atualização a serem propagadas em direção ao nó raiz de acordo com a precisão da estimativa pretendida. Diferentemente do modelo proposto no capítulo 4, os agregados de monitoração representam o conteúdo de um único objeto de gerência e almeja-se apenas a redução do fluxo de informações.

3.2 Monitoração Adaptativa para Máquinas Virtuais

Machida em (MACHIDA; KAWATO; MAENO, 2007) propôs um método de monitoração adaptativa, cujo objetivo consiste no escalonamento ótimo e dinâmico dos recursos gerenciáveis que serão monitorados. Esse ordenamento deve satisfazer a restrições da carga de monitoração máxima e a requisitos de freqüência de atualização das informações. Nesse caso, a adaptação é alcançada em razão da minimização das monitorações concorrentes em um mesmo intervalo. A arquitetura utilizada nessa solução baseia-se em *Web Service Polling Engine* (WSPE) (MACHIDA; KAWATO; MAENO, 2007), cuja finalidade consiste em garantir dados de monitorações recentes por meio do escalonamento dinâmico de atualização das *caches*.

Conforme ilustrado na figura 3.3, a arquitetura de monitoração adaptativa é composta por: uma *cache* de dados, um provedor, um coletor de informações, um contador de *polling* (CP) e um escalonador de atualização. A *cache* de dados armazena informações atualizadas a respeito de cada recurso gerenciável. O provedor é responsável por acessar cada recurso monitorado e disponibilizar as informações ao coletor de informações. O coletor de informações possui funcionamento similar a um processo de monitoração, respeitando o intervalo de *polling* para atualizar a *cache* de dados. O contador de *polling* indica a quantidade de ocorrências de ciclos de *polling* a partir do começo da monitoração para cada recurso alvo. Por fim, tem-se o principal componente dessa arquitetura: o escalonador de atualização. Ele é responsável por, dinamicamente, computar a ordem dos recursos a serem monitorados, definindo um escalonamento.

Para cada recurso monitorado, existe uma estrutura de dados composta por um intervalo CP que provê a quantidade de ciclos de *polling* e um indicador do próximo CP, como pode ser observado na figura 3.4. Quando o próximo CP for atingido pelo contador de *polling* atual, o escalonador de atualização irá disparar requisições ao coletor de informações dizendo quais recursos devem ser monitorados nesse intervalo. No instante de monitoração igual a 6 tem-se dois processos de atualização: *host03* e *vm01*. A garantia do sucesso desse método consiste na distribuição de diferentes valores de próximo CP às aplicações com o objetivo de reduzir os riscos de grandes picos de fluxo de informações de monitoração causados pelas requisições de atualizações. Para isso, foi introduzido um perfil de monitoração que define os limites superiores e inferiores de cada intervalo de atualização para um grupo de recursos. Assim, o administrador precisa apenas gerenciar a alocação de cada recurso através de um de perfil de monitoração ao contrário de gerenciar o seu

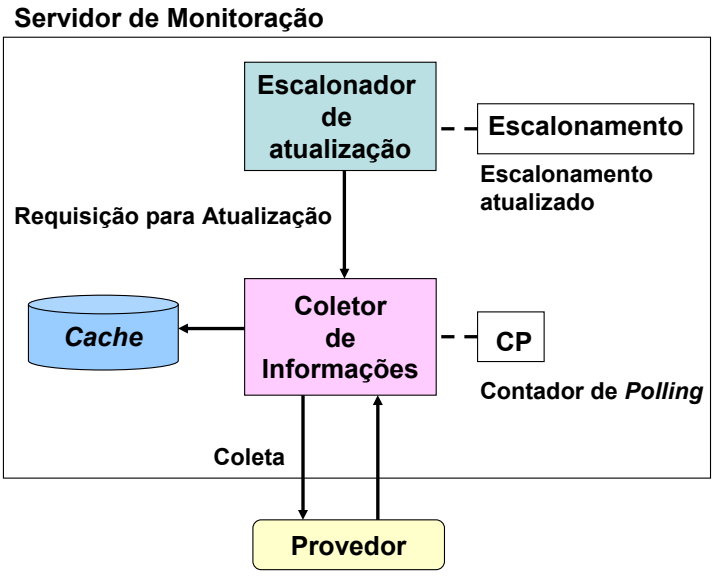


Figura 3.3: Arquitetura de monitoração

tempo de vida. Além do perfil de monitoração, o administrador também deve informar o número máximo de atualizações concorrentes em função da carga de monitoração limite.

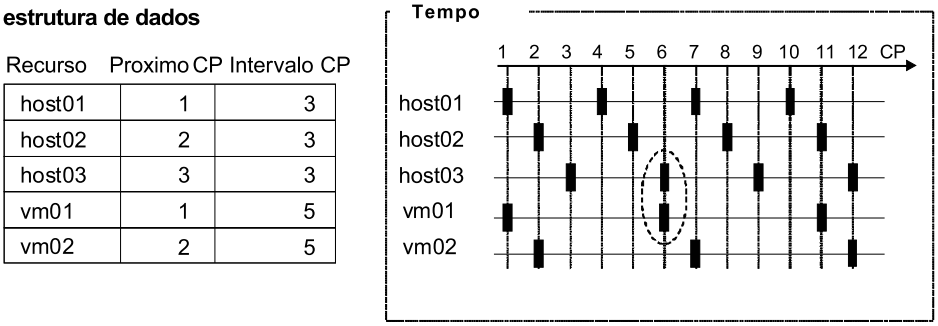


Figura 3.4: Linha do tempo contendo o escalonamento dos processos de atualização

A idéia chave dessa solução consiste em encontrar um escalonamento ótimo dinâmico a partir do perfil de monitoração, o qual satisfaça o intervalo de requisições de atualização para cada recurso e, ao mesmo tempo, minimize o número de atualizações concorrentes. Como esse tipo de problema é classificado como NP-difícil, foi utilizado um algoritmo de aproximação que encontra valores ótimos de escalonamento parciais para cada grupo de recursos possuidor de mesmo intervalo CP.

Para analisar o impacto dessa proposta, foi utilizado o cenário de desfragmentação de máquinas virtuais com o deslocamento delas entre diferentes *hosts* durante 3 minutos. Alcançou-se uma redução de até 80% do tráfego de monitoração, comparando-se os resultados obtidos com a abordagem tradicionalmente empregada. Essa última, por sua vez, caracteriza-se por realizar requisições SNMP, respeitando um intervalo de monitoração, e por não existir restrições quanto ao número de recursos monitorados de forma concor-

rente. No entanto, quando utiliza-se ao máximo as CPUs reduz-se 62% o fluxo de dados de monitoração devido à ocorrência de dispersão do processo de atualização ao longo do tempo. Em síntese, é importante ressaltar que a adaptação nessa abordagem é obtida em razão da minimização das monitorações concorrentes em um mesmo intervalo. Assim como o trabalho anterior, essa abordagem não possui aprendizado e preocupa-se apenas em alternar os distintos conjuntos de informações entre os intervalos de monitoração.

3.3 Promotores e Inibidores Bio-Inspirados para Segurança de Redes Auto-Organizáveis

Dressler (DRESSLER, 2006) propôs um mecanismo adaptativo com a finalidade de reduzir a quantidade de dados monitorados e, assim, ajustar a carga envolvida em sistemas de detecção de ataques. Primeiramente, espera-se monitorar tantos pacotes quantos sejam necessários para obter resultados mais precisos. Todavia, existe um limite superior a respeito da quantidade máxima de pacotes monitorados, o qual é dado pela capacidade de processamento dos sistemas envolvidos. Uma vez que a capacidade de processamento disponível varia conforme o tráfego na rede, criou-se dois tipos de *loops* de resposta inspirados a partir da natureza: promotores e inibidores. Os promotores agem como amplificadores sobre *probes* de monitoração e, assim, tem-se o envio de dados com alta qualidade. Os inibidores, por sua vez, retiram esse efeito causado pelos promotores no caso em que o módulo de detecção de ataques opera na sua capacidade máxima.

Esse mecanismo adaptativo foi introduzido em conjunto com outros dois métodos: configuração direta de filtros sobre *probes* de monitoração e adição de valores de *timeout* para cada entrada da lista negra e da lista branca. Os filtros adicionados aos *probes* de monitoração operam sobre as máquinas que não estão envolvidas em ataques e podem ser usados de modo a reduzir a quantidade de dados a serem processados pelo sistema de detecção de intrusão. A lista negra representa as máquinas que participam de ataques e pode ser utilizada para impedir que pacotes infectados possam alcançar os sistemas de monitoração. Já a lista branca consiste na representação das máquinas que compreendem o tráfego legítimo.

A adaptação, nesse contexto, auto regula-se em razão das medições do tráfego proveniente de ataque e do tráfego legítimo. Por exemplo, se existem recursos disponíveis dentro sistema de detecção de ataque, então tem-se o efeito de amplificação (promotores) através da diminuição dos *timeout* associados à cada entrada da lista negra e lista branca. Analogamente, tem-se o efeito de supressão (inibidores) por meio do aumento do *timeout* em ambas as listas em situações de sobrecarga.

Experimentos realizados demonstram que essa proposta que combina métodos tradicionais (amostragem de pacotes, filtros, listas branca e negra) com promotores e inibidores bio-inspirados propicia a auto-organização completa da monitoração. A adaptação das informações usadas pelo *probe* para detecção de ataques pode ser executada independentemente de cada entidade participante da rede. Contudo, são necessárias a parametrização local e a interoperabilidade entre os elementos vizinhos.

3.4 Alocação Dinâmica de Recursos

Este trabalho consiste no estudo da alocação dinâmica de recursos de servidores HTTP entre múltiplas aplicações. Tesauro (TESAURO, 2007, 2006, 2005) empregou a técnica de aprendizado por reforço para aprender e para adaptar a lista de servidores a serem uti-

lizados para responder a requisições HTTP com o objetivo de minimizar o tempo de resposta. Para isso, foram realizados testes no protótipo de um *Data Center*. Nesse sistema, conforme ilustrado na figura 3.5, cada módulo gerente de aplicação comunica periodicamente a informação de utilidade esperada ($V_x(n_x)$) para o módulo árbitro. A utilidade esperada corresponde a uma função do nível da qualidade de serviço contratada em razão da quantidade de recursos necessários para atender a demanda de requisições HTTP. O módulo árbitro é responsável por computar a alocação ótima de servidores e enviá-la ao gerente sob forma de uma lista de servidores, onde cada um deles está associado a uma aplicação.

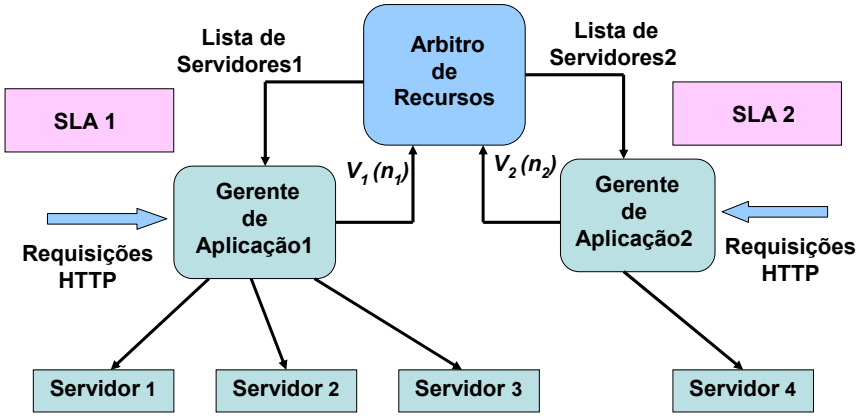


Figura 3.5: Cenário de alocação de recursos do protótipo *Data Center*

O tráfego *web* e a carga de tarefa das aplicações foram emulados através de modelos estocásticos de séries temporais, a fim de variar o tempo de demanda de cada aplicação e, também, de mudar dinamicamente o número de clientes. As decisões de alocações foram realizadas de acordo com uma taxa fixa de 5 segundos de intervalo, na qual cada gerente de aplicação reporta ao árbitro uma curva de utilidade. Essa curva estima valores dentro do limite do nível de serviço acordado entre o cliente e o fornecedor, considerando o estado atual da aplicação em função do número de servidores alocados. Caso o acordo de nível de serviço (*Service Level Agreement* - SLA) não seja cumprido, o cliente pode cobrar multa do fornecedor do serviço.

O gerente da aplicação geralmente recomputa a curva de utilidade de acordo com os estados das aplicações e com as flutuações nos níveis de carga da tarefa. Sobre a curva de utilidade atual, o árbitro encontra a alocação ótima, maximizando o valor total esperado. Inicialmente, o cálculo da utilidade esperada foi realizado por meio da escolha aleatória da alocação dos recursos baseada na política de exploração ϵ-Gulosa (seção 2.2.2), cuja probabilidade é igual a 0.1. Tesauro não varia a probabilidade em seus teste e ainda limita-se ao emprego dessa única política sem compará-la a outras existentes tal qual Boltzmann, por exemplo.

A solução proposta mostrou-se limitante quando aplicada aos cenários em execução, uma vez que não se atingiu a escalabilidade pretendida por requerer muitas iterações. Uma solução híbrida foi empregada, a qual caracteriza-se por adotar o aprendizado por reforço em conjunto com algum conhecimento já existente. Logo, desenvolveu-se uma política inicial e essa foi aprimorada pela execução de um conjunto de treinamento composto por dados reais armazenados. Ou seja, todos os valores de estados, as ações e as recompensas observados em cada aplicação foram gravados e usados no treinamento das

funções valores locais para cada servidor durante o período noturno.

A utilização dessa solução híbrida envolvendo uma política inicial conquistou melhoras significativas no desempenho sobre uma grande variedade de modelos de filas de redes em tempo real. Esse ganho tornou-se ainda maior quando existiram grandes atrasos devido a re-atribuições de servidores.

3.5 Descoberta de Menor Caminho entre Entidades Vizinhas

Uma aplicação baseada no aprendizado por reforço para o balanceamento de carga entre servidores *web*, servidores de *e-mail* e base de dados foi desenvolvida por Shimon Whiteson e Peter Stone (WHITESON; STONE, 2004). Na figura 3.6, tem-se o cenário de aplicação desse sistema, o qual é formado pelos usuários que criam as tarefas a serem executadas e pelos equipamentos representados pelos retângulos hachurados. Cada tarefa subdivide-se em um conjunto de passos e esses, por sua vez, podem exigir diferentes recursos de execução como, por exemplo, servidores *web* contendo distintas aplicações. Dessa forma, cada tarefa gerada pode percorrer diversos equipamentos até completar sua execução e retornar ao seu ponto de partida. As linhas desenhadas entre esses componentes correspondem aos caminhos existentes, os quais podem compor uma rota.

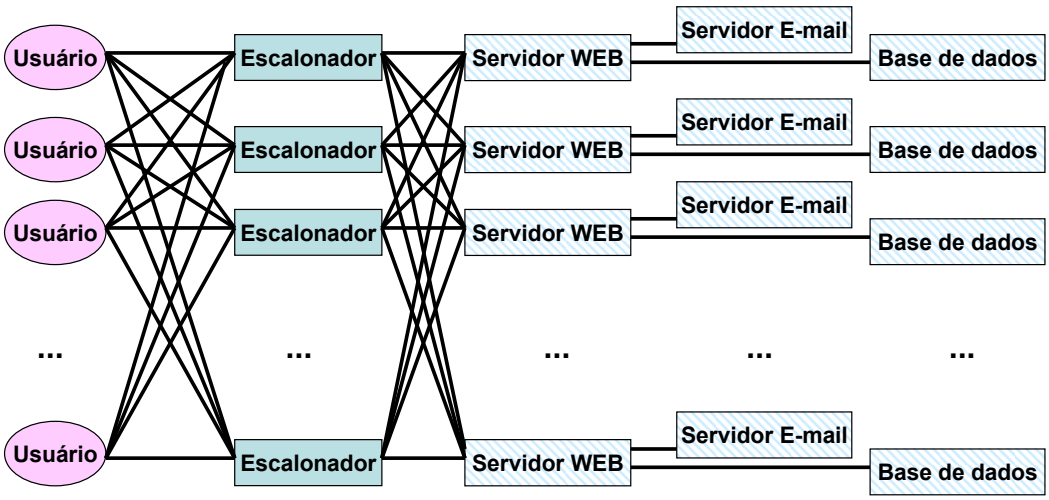


Figura 3.6: Balançamento de carga entre servidores

Nesse sistema, a primeira etapa consiste no encaminhamento da tarefa criada pelo usuário para o escalonador, o qual fornece a melhor rota de recursos para execução de todos os passos pertencentes a essa tarefa. A melhor rota é obtida em função da maximização dos pesos das tarefas que são compostos pela prioridade de execução diferenciada a cada usuário e pelo valor associado a cada tipo de tarefa. Nesse contexto, o aprendizado por reforço é utilizado para automatizar o aprendizado das políticas de encaminhamento do sistema e, assim, obter uma função de valor de rotas. Essa função valor de rotas estima o menor tempo gasto para execução de uma tarefa partindo de um nodo origem para um nodo destino específico, considerando todos os nodos vizinhos disponíveis.

O algoritmo empregado na descoberta das rotas é o *Q-routing*, o qual caracteriza-se pela execução em tempo-real. O algoritmo *Q-routing* utiliza uma tabela Q que fornece estimativas de tempo gasto por um passo da tarefa dado um nodo origem e um nodo escolhido como próximo destino. Uma característica peculiar dessa solução consiste na

presença de módulos de aprendizado por reforço em cada nodo da rede. Assim, cada nodo é capaz de aprender qual o nodo vizinho mais apropriado para executar o próximo passo de uma tarefa e, ao mesmo tempo, alcançar mais rapidamente o nodo destino. Para finalizar, uma heurística de inserção de tarefas no escalonador foi criada, na qual as tarefas que chegam a ele são colocadas nas posições da fila com mais alto valor estimado.

Esse trabalho realizou um comparativo das rotas aprendidas antes e depois da adoção da heurística de inserção de tarefas e, ainda, considerou outras estratégias de heurísticas fixas as quais são tradicionalmente empregadas para solução dessa questão. No entanto, não foram explorados outros algoritmos de aprendizado por reforço como, por exemplo, Boltzmann. Por meio do resultado obtido, constatou-se que, envolvendo progressivamente ambientes de redes mais complexos, os encaminhamentos aprendidos e as políticas de escalonamento obtêm melhoras significativas em relação às políticas baseadas em heurísticas.

3.6 Considerações Parciais

Nessa seção foram apresentados dois subconjuntos de trabalhos distintos. O primeiro conjunto possui foco na diminuição da quantidade de informações monitoradas por meio de diferentes formas de adaptação (protocolo, escalonador de atualização, reconfiguração de parâmetros por promotores e inibidores). O segundo grupo está centrado no uso de aprendizado por reforço para que os sistemas em questão aprendam a se adaptar conforme as condições observadas no ambiente.

O trabalho de Gonzalez e Stadler (seção 3.1) propôs o protocolo A-GAP que objetiva diminuir a quantidade de mensagens enviadas dos nós filhos em direção ao pai pela introdução de filtros com estimativas dinâmicas de erro. Contudo, a adaptação alcançada por essa abordagem ficou restrita à seleção das mensagens de atualização a serem enviadas em direção ao nó raiz de acordo com a precisão da estimativa pretendida. Outro detalhe importante consiste no fato dos agregados de monitoração representarem apenas o conteúdo de um único objeto de gerência.

No artigo de Machida (seção 3.2) foi apresentada outra abordagem para a redução da quantidade de informações de monitoração. O uso de um escalonador de atualizações decide quais conjuntos de dados serão monitorados em um dado instante. Cada conjunto de dados somente é modificado quando altera-se uma configuração prévia ou no caso de ativação/desativação de uma máquina virtual. Assim como o trabalho anterior, essa abordagem não possui aprendizado e preocupa-se apenas em alternar os distintos conjuntos de informações entre os intervalos de monitoração. Não foi objetivo desse trabalho a adaptação das variáveis pertencentes a um mesmo conjunto de monitoração.

Na seção 3.3, a abordagem inspirada na biologia através de promotores e inibidores foi descrita. Essa abordagem foi a mais complexa, pois combina a técnica de amostragem de pacotes, compressão de dados, classificação em lista branca e negra para reduzir a quantidade de pacotes monitorados. A adaptação desse modelo correspondeu ao processo de auto regulação (aumento/diminuição) do tempo de *timeout* associado às listas branca e negra, baseado no comportamento do tráfego da rede e na capacidade de processamento disponível nos recursos do sistema.

A partir da seção 3.4, foram introduzidos trabalhos que lançam mão de aprendizado por reforço para resolver os problemas de alocação dinâmica de recursos e de descoberta de menor caminho entre entidades vizinhas. Ambos os trabalhos fazem alusão à grande quantidade de iterações no processo de aprendizagem, conseqüentemente à morosidade,

para a obtenção de resultados expressivos. Para resolver esse problema, elas empregam respectivamente políticas iniciais e heurísticas. As abordagens propostas foram comparadas com métodos tradicionalmente empregados na solução dessa natureza de problemas, ignorando outros algoritmos relevantes propostos dentro da área de aprendizado por reforço.

A luz dos trabalhos apresentados, existem iniciativas consistentes em pesquisas para prover aprendizado e adaptação. Contudo, esses estudos não contemplam o cenário de monitoração de redes e, ainda, possuem objetivos distintos daqueles explorados no capítulo 5. Além disso, mesmo os trabalhos com ênfase na investigação da abordagem de aprendizado por reforço, têm limitações no sentido de avaliações mais exaustivas. No próximo capítulo é descrita uma proposta que tenta adaptar o conjunto de informações de maneira a ampliar ou restringir o escopo de monitoração e, assim, propiciar o acompanhamento e a observação mais precisa.

4 MONITORAÇÃO ADAPTATIVA

Conforme exposto anteriormente, a evolução dos sistemas de computação está direcionando para soluções cujo princípio fundamental de desenvolvimento é o auto-gerenciamento (seção 2.1). Essa característica é responsável por conferir aos sistemas maior independência em relação à intervenção manual de administradores. Para que isso aconteça, o sistema deve ser capaz de se adaptar frente às alterações ocorridas no ambiente sem transparecer ao usuário. Uma oportunidade de aplicação desse conceito de auto-gerência é vislumbrada nos processos de monitoração de redes, de serviços e de aplicações.

Dentro do escopo de redes, a disciplina de monitoração é, tipicamente, realizada com um conjunto fixo e pré-definido de objetos gerenciáveis escolhidos em função do ambiente a ser analisado (por exemplo, interfaces de dispositivos de rede). No entanto, o especialista que seleciona esse grupo de objetos gerenciáveis raramente realiza alterações, tal como inclusão de novos objetos a serem monitorados, e acaba trabalhando com um conjunto estático. A falta de manutenção nesse conjunto de informações de monitoração torna-se um empecilho para a obtenção de indicadores, os quais possam contribuir para melhorar a compreensão do estado funcional da rede. Sendo assim, a adaptação dinâmica do conjunto de objetos gerenciáveis representa um instrumento importante para conferir flexibilidade e assegurar evolução de processos de monitoração.

Este trabalho propõe um modelo para suporte à monitoração adaptativa de redes, serviços e aplicações, capaz de ampliar e restringir o conjunto de objetos gerenciáveis para permitir o acompanhamento e a observação mais precisa. Sobre esse modelo, aplica-se a técnica de aprendizado por reforço, utilizando o algoritmo *Q-Learning* (seção 2.2.2) combinado com três políticas distintas (ϵ-Gulosa, *Boltzmann* e Bloqueio Temporário). A última delas é proposta no contexto desse trabalho em busca de um desempenho mais favorável ao domínio do problema, representando uma contribuição adicional significativa desta dissertação.

Este capítulo está organizado da seguinte forma: a seção 4.1 introduz o o modelo elaborado a partir da abordagem de aprendizado por reforço descrita na seção 2.2 e a seção 4.2 ilustra o modelo em execução em um cenário de aplicação.

4.1 Modelo para Monitoração Adaptativa

O modelo parte da representação de estados de monitoração, que são interligados formando uma árvore. A cada estado são associados objetos a serem monitorados. A figura 4.1 ilustra um exemplo de árvore de estados. O nó posicionado na raiz consiste no estado inicial; nós filhos do nó raiz representam estados para os quais o processo de monitoração pode evoluir a partir do inicial e, assim, sucessivamente. Observe que, como regra geral, os estados filhos, além de incluírem um ou mais novos objetos, herdam

aqueles discriminados no nodo pai. Por exemplo, o estado S_1 é composto pelos objetos gerenciáveis A e B, sendo o primeiro herdado do estado pai S_0. São considerados nós terminais os estados cujo sistema não pode mais evoluir em uma das direções. Portanto, nós raiz e folhas são ditos terminais.

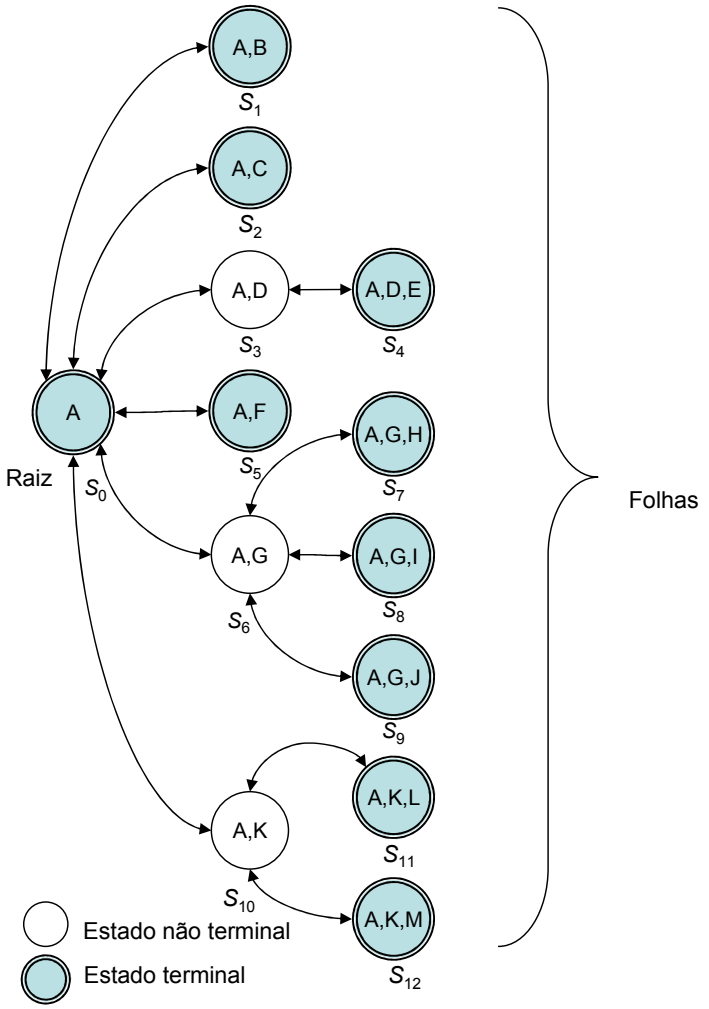


Figura 4.1: Árvore de Estados

Esta árvore de estados é base para o processo adaptativo de monitoração. O movimento inicial do sistema é realizado a partir do nodo raiz, o qual apresenta o menor conjunto de informações de monitoração. Conforme percorre-se a árvore na direção das folhas, amplia-se a quantidade de indicadores que, potencialmente, podem contribuir para análises mais precisas. As transições entre os estados são permitidas em duas direções: do nodo raiz para os nodos folhas (expansão) e dos nodos folhas para a raiz (contração). No caso específico dos nós raiz e folhas, por eles serem terminais, a transição somente é possível em um único sentido.

Como pode ser observado na figura 4.2, cada estado da árvore tem associado a si as seguintes informações:

- *Nome*: identificação única do estado;

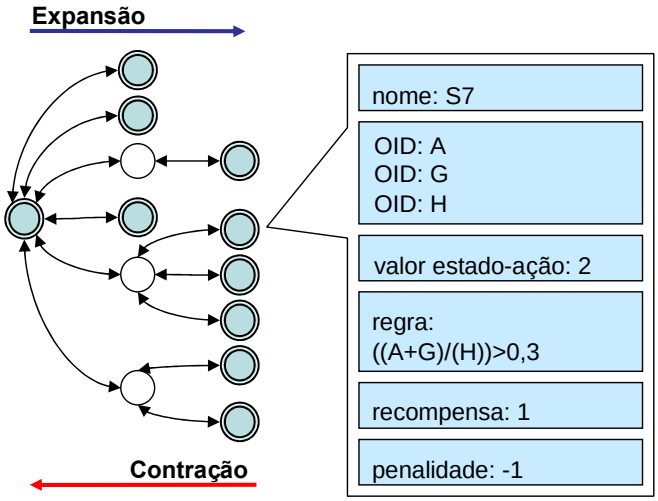


Figura 4.2: *Zoom* no Estados

- *OIDs*: identificadores únicos dos objetos gerenciáveis. (ex: *A* na figura 4.2 poderia ser a variável `ipInReceives`, cujo *OID* é :`.1.3.6.1.2.1.4.3.0`);
- *valor estado-ação*: representa o valor acumulado pela visitação desse estado por transições passadas (a ser detalhado mais a frente);
- *regra*: expressão matemática que, uma vez satisfeita, levará o sistema à expansão em direção aos estados folhas (ou à contração, caso contrário);
- *recompensa*: importância a ser acrescida ao valor estado-ação quando a regra é satisfeita. Então, aplica-se a ação de expansão sobre o estado;
- *penalidade*: importância a ser decrementada do valor estado-ação quando a regra não é satisfeita. Logo, o estado recebe a ação de contração.

Conforme visto na seção 2.2.1, a abordagem de aprendizado por reforço apresenta somente uma função de recompensa que está associada ao estado e a ação a ser aplicada. Essa recompensa pode ser tanto positiva quanto negativa. Fazendo uma alusão ao modelo proposto, observe que tem-se apenas duas ações possíveis sobre um estado: a expansão e a contração. Nesse caso, tanto a *recompensa* quanto a *penalidade* possuem o mesmo significado semântico da função de recompensa utilizada pelo aprendizado por reforço, uma vez que estão associadas a ações distintas aplicadas sobre o mesmo estado.

O funcionamento do modelo de suporte à monitoração adaptativa proposto é expresso pelo algoritmo descrito pelo pseudocódigo apresentado na página seguinte. Observe que têm-se como parâmetros de entradas do algoritmo “AvaliaNodo” uma referência para o estado no qual se encontra o sistema e o intervalo de amostragem considerado.

O algoritmo inicia pela recuperação dos valores dos objetos que são discriminados no estado corrente, conforme mostrado na figura 4.2. Essa monitoração é realizada no intervalo de tempo pré-definido, no qual os objetos são amostrados no início e no fim do período. Essas duas amostras (t, t+1) obtidas são imprescindíveis às regras que possuem cálculos baseados na diferença (Δ) de seus valores nesse espaço de tempo. Um exemplo em que a subtração é tipicamente empregada corresponde à utilização de variáveis do

Algoritmo 2 AvaliaNodo

```
1:  AvaliaNodo(Estado s, int intervaloAmostra){
2:  //obtém os atributos do estado s
3:  Obtem_Objetos_Nodo(s);
4:
5:  //obtém dados monitorados no início e no fim do periodo
6:  dados_Monitorados(t,t+1) = Coleta(oids(s), intervaloAmostra);
7:
8:  result = Avalia_Expressão(regra(s), oids(s), dados_Monitorados(t,t+1)));
9:
10: Se (result e ((terminal e primeira_vez) ou (!terminal))) então {
11:    valorEstado_Ação(s) = (1 − α) * valorEstado_Ação(s) + α * recompensa(s);
12:    próximoNodo = Obtem_Proximo_ValorEstado_Ação(filhos);
13:    }
14: Senão Se  (((terminal e primeira_vez) ou (!terminal)))  então {
15:    valorEstado_Ação(s) = (1 − α) * valorEstado_Ação(s) - α * penalidade(s);
16:    próximoNodo = nodoPai(s);
17:    }
18: Senão
19:    próximoNodo = s;
20:
21: retorna  proximoNodo;
```

tipo *counter*. Para aquelas que usualmente consideram suficientes amostras instantâneas de valores como, por exemplo, variáveis do tipo *gauge*, apenas a segunda informação coletada poderá ser utilizada na avaliação da regra. É importante ressaltar que o operador possui liberdade de utilizar qualquer uma das duas abordagens anteriormente apresentadas de forma a refletir a intenção de monitoração para a especificação da regra.

De posse dos dados monitorados em t e $t + 1$ (linha 6), o próximo passo consiste em avaliar a expressão matemática que descreve a relação entre as variáveis desse estado "s". Se essa regra é satisfeita, por exemplo, *((A+G)/(H))>0.3*, então atualiza-se o *valor estado-ação* através da sua multiplicação pelo termo *(1-*α*)* somado à *recompensa* associada ao estado analisado, multiplicada por α (linha 11). Note que foi introduzido um fator α de correção da atualização do *valor estado-ação*, o qual corresponde a uma taxa de aprendizado normalmente definida entre 0 e 1. O seu emprego objetiva impedir que os estados possuidores de uma baixa *recompensa* e que sejam freqüentemente visitados contenham *valores estado-ação* maiores do que estados menos visitados ou nunca alcançados cujos reforços sejam maiores. Por exemplo, considere um estado A com recompensa 0.1 e que foi visitado 10 mil vezes, e o estado B com recompensa 100 e que foi visitado apenas uma vez. Nesse caso, o estado A vai parecer melhor do que o B se simplesmente ocorrer a soma da recompensa ao *valor estado-ação* sem utilizar o fator de correção, ficando igual a 100 para o estado A e 90 para o estado B. Com o uso do fator de correção α igual a 0.1, por exemplo, o *valor estado-ação* do estado A fica igual a 6,3 e o *valor estado-ação* de B igual a 9. Como uma última observação, é valido destacar que as recompensas podem ser distintas entre os diferentes nós que compõem a árvore de estados.

Além da avaliação da regra, obtém-se como o nodo sucessor para a expansão (linha 12) o estado filho levando-se em conta três processos de seleção distintos: ϵ-Guloso,

Boltzmann e Bloqueio Temporário do estado. O método ϵ-Guloso, como visto na seção 2.2.2, na maioria dos casos, escolhe o estado filho que possui o maior *valor estado-ação*. No entanto, existe uma pequena probabilidade de selecionar outros estados que apresentam piores *valores estado-ação*, determinada por uma variável denominada taxa de exploração.

A seleção de *Boltzmann* também envolve probabilidade, porém considera os valores estado-ação de maneira relativa, ou seja, a probabilidade da escolha do próximo estado depende da comparação de seu valor estado-ação em relação aos demais estados filhos. No caso de dois estados possuírem *valores estado-ação* elevados, por exemplo 50 e 45, ambos terão a mesma probabilidade de seleção, uma vez que essa é definida pela variável denominada temperatura. Entretanto, se existe um único *valor estado-ação* que é muito elevado em relação aos demais, ele provavelmente será escolhido.

A última opção corresponde ao bloqueio temporário do estado penalizado, contribuição importante desse trabalho. Nessa abordagem, a escolha do próximo estado é realizada de maneira gulosa, ou seja, priorizando a seleção do estado com maior *valor estado-ação*, considerando que ele não foi penalizado em um passado recente. Se o estado foi recém penalizado, então ele estará bloqueado temporariamente e não será considerado pelo processo de seleção, mesmo que ele apresente o maior *valor estado-ação*. Sugere-se que o *tempo de suspensão* deva considerar a quantidade de estados, o tempo transcorrido no processo de aquisição dos objetos monitorados e o tempo de processamento da avaliação da *regra*, para que se possa explorar outros estados vizinhos.

Retornando ao algoritmo, caso a *regra* não seja satisfeita, deve-se atualizar o *valor estado-ação* por meio da sua multiplicação pelo termo *(1-*α*)* somado à *penalidade*, multiplicada por α (linha 15). Isso implica na transição de retorno para o seu estado pai, como mostrado na linha 16. Da mesma forma que a *recompensa*, a *penalidade* também é diferenciada em cada nó. Observe que as linhas 11 e 15 do algoritmo correspondem à função de estimação Q (2.4) apresentada na seção 2.2.2. Por outro lado, existe uma pequena diferença em relação ao uso do fator γ, uma vez que ele é considerado igual a zero nesse modelo. Dessa maneira, pretende-se descartar a influência dos valores dos estados futuros no cálculo do *valor estado-ação* analisado.

Quando o nó raiz for atingido pela contração ou quando algum dos nodos folhas for alcançado pela expansão, essas duas situações indicam que não existe mais a opção de transição nas respectivas direções nesses estados. Se a condição exigida pela regra persiste sendo satisfeita nos estados folhas ou permanece não sendo atingida no nodo raiz, então não haverá nenhuma transição e continua-se no mesmo estado (linha 19). Nessa situação, evita-se que o *valor estado-ação* seja constantemente recompensado ou penalizado após a primeira atualização, uma vez que isso dificulta o processo de aprendizado devido à elevação contínua e desproporcional do seu valor. Isso é realizado por meio do acréscimo ao algoritmo de um identificador de *detecção de loop*. Uma nova atualização do *valor estado-ação* é realizada somente quando o sistema retorna a evoluir e sai desse nodo terminal. Para isso, a regra de avaliação não deve ser mais satisfeita e, assim, o sistema retorna para o estado pai.

Uma característica peculiar desse modelo consiste na existência de uma lacuna de tempo entre os ciclos de obtenção dos valores correspondentes às variáveis, a qual é desprezada pela análise. Essa lacuna, representada na figura 4.3 pelo ω_i, é composta pelo tempo de processamento da avaliação do estado a partir dos dados coletados até a obtenção dos valores monitorados do próximo estado S_i. Normalmente, ω deve ser muito pequeno, ficando restrito à ordem de grandeza de milissegundos, não ocasionando ne-

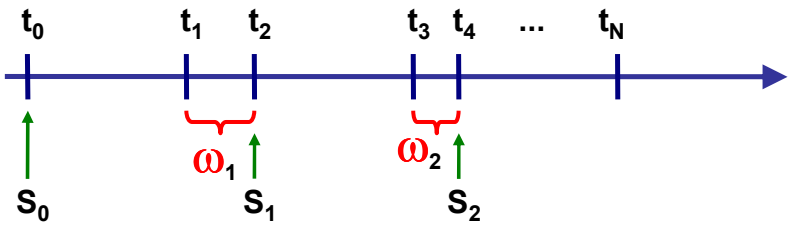


Figura 4.3: Linha do tempo da Monitoração

nhum impacto no processo de amostragem se o mesmo for desprezado. O procedimento de desconsiderar as variações dos valores dos objetos ocorridas nesse ínterim gera uma descontinuidade no espaço amostral de informações. No entanto, o objetivo desse procedimento consiste em preservar a mesma relação temporal entre os valores das variáveis obtidas no ciclo anterior e seus novos valores no período atual.

Em síntese, esse modelo de suporte à monitoração baseia-se no acúmulo de experiências obtidas por meio de diversas ações que podem ser bem ou mal sucedidas. O processo de escolha do próximo estado a ser alcançado é empírico, uma vez que baseia-se no método de tentativa e erro proposto pela metodologia de aprendizado por reforço. Assim, a partir de cada iteração de monitoração, é possível adaptar o sistema à dinâmica do ambiente de análise e, ao mesmo tempo, aprender padrões de monitoração. Essa adaptação de indicadores tem potencial para reduzir a necessidade da intervenção manual do operador no processo de monitoração das redes de computadores, como será demonstrado na próxima seção.

4.2 Exemplo de Aplicação

Considerando o modelo anteriormente exposto, apresenta-se, agora, um exemplo de sua instanciação. O exemplo consiste na monitoração de um dispositivo com capacidade de roteamento. A monitoração desse tipo de equipamento permite identificar diversas situações indesejáveis, tais como corrompimento ou desatualização na tabela de rotas, erros de endereçamento e *buffers* com capacidade de armazenamento esgotada. Para tal, diversos indicadores precisam ser combinados para uma caracterização precisa da situação em curso. A título de ilustração, neste exemplo são exploradas variáveis da MIB II (MCCLOGHRIE; ROSE, 1991), tais como `icmpInMsgs`, `icmpInErrors`, `icmpInEchoReps`, `ipForwarding`, `ipFragFails`,`ipInAddrErrors`, `ipInDiscards`, `ipInHdrErrors`, `ipInReceives`, `ipOutDiscards`, `ipOutNoRoutes`, `ipOutRequests`, `ipReasmFails`, `ipReasmReqds`. Essas variáveis foram selecionadas a fim de constituir um estudo de caso relativo a investigação da presença de um distúrbio na rede analisada.

Seguindo a mesma estrutura organizacional da árvore presente na figura 4.1, dividiu-se o conjunto de variáveis em 13 estados (S_0 a S_{12}) de acordo com o potencial grau de afinidade existente entre elas a fim prover subsídios para elucidar possíveis tipos de problemas ocorridos na rede. De maneira geral, as informações presentes nesses indicadores auxiliam na caracterização do tráfego de datagramas IP em relação tanto à interface de entrada quanto à de saída do dispositivo analisado. Considerado o estado inicial da árvore, uma vez que obtém-se uma baixa ou uma alta quantia de datagramas IP recebidos, deve-se investigar qual a razão do possível distúrbio da rede. De acordo com estados terminais,

com exceção da raiz, pode-se constatar a presença de uma elevada quantidade:

- (*i*) de erros de endereçamento em datagramas IP recebidos (S_0,S_2);
- (*ii*) de erros no cabeçalho de datagramas IP recebidos (S_0,S_1);
- (*iii*) de falhas provenientes do processo de remontagem de datagramas IP fragmentados (S_0,S_3,S_4);
- (*iv*) de descarte de datagramas IP recebidos (S_0,S_5) e enviados (S_0,S_6,S_7), os quais não apresentam qualquer tipo de falha. Esse descarte possui a finalidade de evitar o processamento dos datagramas IP devido a falta de espaço de armazenamento nos *buffer* de entrada e de saída, por exemplo;
- (*v*) de datagramas IP não encaminhados para seus destinos devido a não existência de rotas (S_0,S_6,S_8);
- (*vi*) de datagramas IP não enviados para seus destinos, uma vez que eles precisam ser fragmentados, mas não podem ser pois existe uma *flag* ligada no pacote que impossibilita seu particionamento (S_0,S_6,S_9);
- (*vii*) de mensagens ICMP recebidas que apresentam erros relacionados ao protocolo (por exemplo, erro de *checksum* no datagrama ICMP) (S_0,S_{10},S_{11});
- (*viii*) de mensagens ICMP ECHO Replay recebidas (S_0,S_{10},S_{12}).

Pegando como exemplo as informações associadas ao estado S_1 na figura 4.4 (a), observa-se que o referido estado é composto pelas variáveis *ipInAddrErrors* e *ipInReceives*. A *regra* de avaliação utilizada nesse nodo emprega a subtração dos valores amostrados no início e no fim do intervalo da monitoração, sendo indicado pelo símbolo *delta* (Δ). Essa inequação, para ser satisfeita, necessita que 30% do total de datagramas recebidos sejam descartados, em razão de apresentarem endereços de destino inválidos. A *recompensa* e a *penalidade* associadas ao estado são 2 e -1, respectivamente. Uma consideração importante consiste no valor das variáveis `ipInReceives` e `ipOutRequests`, cuja significância varia conforme o tipo da rede e a função exercida pelo dispositivo com capacidade de roteamento como, por exemplo, um roteador de borda. Nesse trabalho, foi utilizado o valor igual 1000 a título de ilustração.

Ainda antes de instanciar o exemplo, considere a tabela de amostras presente na figura 4.4 (b), na qual cada instante é representado por t_x, com espaçamento de 30 segundos, que corresponde ao intervalo de monitoração. Como pode ser observado, nem todas as variáveis possuem os valores informados em todos os instantes (t_0 a t_{11}), uma vez que as monitorações realizadas são compostas apenas pelos `OIDs` que fazem parte do estado analisado na iteração do sistema.

Assumindo hipoteticamente variações ocorridas nos valores dos objetos de acordo com a tabela da figura 4.4 (b), tem-se a evolução do sistema a partir do estado S_0. É importante ressaltar que na instanciação desse exemplo, não importa o tipo da abordagem empregada na seleção do próximo estado, pois todas elas apresentam o mesmo comportamento quando os estados são visitados pela primeira vez. Para melhor visualização, foi reduzida a representação da árvore da figura 4.1 aos estados utilizados neste exemplo.

Tem-se como primeiro passo do sistema, a coleta dos dados de S_0 em t_0 (1000) e t_1 (2005) a fim de se obter a variação nesse período. Logo, subtrai-se o valor amostrado em

Estado	OID	Regra	*R.[a]	*P.[b]
S_0	ipInReceives ipOutRequests	(ΔipInReceives >= 1000) || (ΔipOutRequests >= 1000)	2	-1
S_1	ipInAddrErrors ipInReceives	$\left(\frac{\Delta\text{ipInAddrErrors}}{\Delta\text{ipInReceives}}\right) > 30\%$	2	-1
S_2	ipInHdrErrors ipInReceives	$\left(\frac{\Delta\text{ipInHdrErrors}}{\Delta\text{ipInReceives}}\right) > 30\%$	2	-1
S_3	ipReasmReqds ipInReceives	$\left(\frac{\Delta\text{ipReasmReqds}}{\Delta\text{ipInReceives}}\right) > 30\%$	2	-1
S_4	ipReasmFails ipReasmReqds ipInReceives	$\left(\left(\frac{\Delta\text{ipReasmFails}}{\Delta\text{ipReasmReqds}}\right) > 30\%\right.$	2	-1
S_5	ipInDiscards ipInReceives	$\left(\frac{\Delta\text{ipInDiscards}}{\Delta\text{ipInReceives}}\right) > 30\%$	2	-1
S_6	ipForwarding ipOutRequests ipInReceives	$\left(\left(\frac{\Delta\text{ipForwarding}}{\Delta\text{ipInReceives}}\right) > 30\%\right)$ && (ΔipOutRequests > 1000)	2	-1
S_7	ipOutDiscards ipForwarding ipOutRequests ipInReceives	$\left(\frac{\Delta\text{ipOutDiscards}}{\Delta\text{ipForwarding}+\Delta\text{ipOutRequests}}\right) > 30\%$	2	-1
S_8	ipOutNoRoutes ipForwarding ipOutRequests ipInReceives	$\left(\frac{\Delta\text{ipOutNoRoutes}}{\Delta\text{ipForwarding}+\Delta\text{ipOutRequests}}\right) > 30\%$	2	-1
S_9	ipFragFails ipForwarding ipOutRequests ipInReceives	$\left(\frac{\Delta\text{ipFragFails}}{\Delta\text{ipForwarding}+\Delta\text{ipOutRequests}}\right) > 30\%$	2	-1
S_{10}	icmpInMsgs ipInReceives	$\left(\frac{\Delta\text{icmpInMsgs}}{\Delta\text{ipInReceives}}\right) > 30\%$	2	-1
S_{11}	icmpInErrors icmpInMsgs ipInReceives	$\left(\frac{\Delta\text{icmpInErrors}}{\Delta\text{icmpInMsgs}}\right) > 30\%$	2	-1
S_{12}	icmpInEchoReps icmpInMsgs ipInReceives	$\left(\frac{\Delta\text{icmpInEchoReps}}{\Delta\text{icmpInMsgs}}\right) > 30\%$	2	-1

(a)

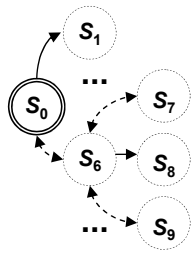
[a]Recompensa
[b]Penalidade

OID	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
icmpInMsgs	-	-	-	-	-	-	-	-	-	-	-	-
icmpInErros	-	-	-	-	-	-	-	-	-	-	-	-
icmpInEchoReps	-	-	-	-	-	-	-	-	-	-	-	-
ipForwarding	-	-	-	-	-	-	2245	3062	3324	3817	4064	4296
ipFragFails	-	-	-	-	-	-	-	-	-	-	-	-
ipInAddrErrors	-	-	123	188	-	-	-	-	-	-	-	-
ipInDiscards	-	-	-	-	-	-	-	-	-	-	-	-
ipInHdrErrors	-	-	-	-	-	-	-	-	-	-	-	-
ipInReceives	1000	2005	3026	4043	5061	6078	7085	8087	9090	10005	11000	11902
ipOutDiscards	-	-	-	-	-	-	-	-	-	-	-	-
ipOutNoRoutes	-	-	-	-	-	-	-	-	1036	1723	2521	3096
ipOutRequests	-	-	-	-	-	-	4078	5236	6142	7371	8529	9852
ipReasmReqds	-	-	-	-	-	-	-	-	-	-	-	-
ipReasmFails	-	-	-	-	-	-	-	-	-	-	-	-

(b)

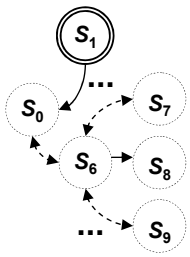
Figura 4.4: Instanciação de um exemplo de aplicação

t_1 do valor amostrado em t_0 e aplica-se o resultado obtido (1005) na regra de avaliação pertencente ao estado S_0. Note que ocorre a satisfação da regra (1005 > 1000 = verdadeiro), o que implica em somar a *recompensa* (2), multiplicada pela taxa de aprendizado α (0.2), ao *valor estado-ação* de S_0, o qual é multiplicado por um menos a taxa de aprendizado α (0.2), resultando 0.4 (figura 4.5 (a)). Como todos os *valores estado-ação* dos



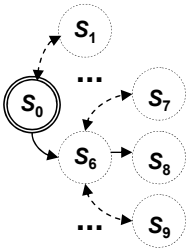
	$t_{0,1}$	$t_{2,3}$	$t_{4,5}$	$t_{6,7}$	$t_{8,9}$	$t_{10,11}$
S_0	0.4					
S_1	0					
S_2	0					
S_3	0					
S_4	0					
S_5	0					
S_6	0					
S_7	0					
S_8	0					
S_9	0					
S_{10}	0					
S_{11}	0					
S_{12}	0					

(a)



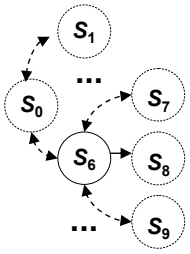
	$t_{0,1}$	$t_{2,3}$	$t_{4,5}$	$t_{6,7}$	$t_{8,9}$	$t_{10,11}$
S_0	0.4	0.4				
S_1	0	-0.2				
S_2	0	0				
S_3	0	0				
S_4	0	0				
S_5	0	0				
S_6	0	0				
S_7	0	0				
S_8	0	0				
S_9	0	0				
S_{10}	0	0				
S_{11}	0	0				
S_{12}	0	0				

(b)



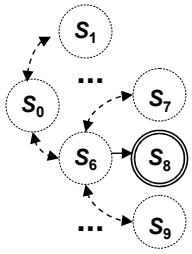
	$t_{0,1}$	$t_{2,3}$	$t_{4,5}$	$t_{6,7}$	$t_{8,9}$	$t_{10,11}$
S_0	0.4	0.4	0.72			
S_1	0	-0.2	-0.2			
S_2	0	0	0			
S_3	0	0	0			
S_4	0	0	0			
S_5	0	0	0			
S_6	0	0	0			
S_7	0	0	0			
S_8	0	0	0			
S_9	0	0	0			
S_{10}	0	0	0			
S_{11}	0	0	0			
S_{12}	0	0	0			

(c)



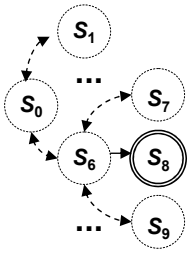
	$t_{0,1}$	$t_{2,3}$	$t_{4,5}$	$t_{6,7}$	$t_{8,9}$	$t_{10,11}$
S_0	0.4	0.4	0.72	0.72		
S_1	0	-0.2	-0.2	-0.2		
S_2	0	0	0	0		
S_3	0	0	0	0		
S_4	0	0	0	0		
S_5	0	0	0	0		
S_6	0	0	0	0.4		
S_7	0	0	0	0		
S_8	0	0	0	0		
S_9	0	0	0	0		
S_{10}	0	0	0	0		
S_{11}	0	0	0	0		
S_{12}	0	0	0	0		

(d)



	$t_{0,1}$	$t_{2,3}$	$t_{4,5}$	$t_{6,7}$	$t_{8,9}$	$t_{10,11}$
S_0	0.4	0.4	0.72	0.72	0.72	
S_1	0	-0.2	-0.2	-0.2	-0.2	
S_2	0	0	0	0	0	
S_3	0	0	0	0	0	
S_4	0	0	0	0	0	
S_5	0	0	0	0	0	
S_6	0	0	0	0.4	0.4	
S_7	0	0	0	0	0	
S_8	0	0	0	0	0.4	
S_9	0	0	0	0	0	
S_{10}	0	0	0	0	0	
S_{11}	0	0	0	0	0	
S_{12}	0	0	0	0	0	

(e)



	$t_{0,1}$	$t_{2,3}$	$t_{4,5}$	$t_{6,7}$	$t_{8,9}$	$t_{10,11}$
S_0	0.4	0.4	0.72	0.72	0.72	0.72
S_1	0	-0.2	-0.2	-0.2	-0.2	-0.2
S_2	0	0	0	0	0	0
S_3	0	0	0	0	0	0
S_4	0	0	0	0	0	0
S_5	0	0	0	0	0	0
S_6	0	0	0	0.4	0.4	0.4
S_7	0	0	0	0	0	0
S_8	0	0	0	0	0.4	0.4
S_9	0	0	0	0	0	0
S_{10}	0	0	0	0	0	0
S_{11}	0	0	0	0	0	0
S_{12}	0	0	0	0	0	0

(f)

Figura 4.5: Evolução do sistema

nodos S_1, S_2, S_4, S_5, S_6 e S_{10} possuem inicialmente o mesmo valor igual a zero, então, deve-se escolher aleatoriamente um deles. Nessa iteração, o estado S_1 foi selecionado para ser a primeira tentativa.

Observe que o sistema apenas monitora as variáveis pertencentes ao estado corrente, nesse caso S_1. Contudo, a diferença dos dados obtidos em t_3 e t_2 (188-123=65), quando aplicada à *regra* de S_1 (0,65% > 30% = falso), revela que não ocorre a satisfação da equação. Nesse caso, decrementa-se do *valor estado-ação*α (0.2), o qual é multiplicado por um menos a taxa de aprendizado α (0.2), a *penalidade* associada ao S_1, multiplicada pela taxa de aprendizado α igual a 0.2, resultando em -0.2 (figura 4.5 (b)), e retorna-se ao estado anterior S_0. Logo, a escolha dessa tentativa não produziu uma boa experiência ao sistema.

De volta ao estado S_0, nota-se que a variação do seu valor no período $t_{4,5}$ continua satisfazendo a *regra* de avaliação. Isso implica em somar novamente a *recompensa* (2), multiplicada pela taxa de aprendizado α, ao valor *estado-ação*, cujo resultado fica igual a 0.72 (figura 4.5 (c)). Como S_1 foi penalizado anteriormente, os estado de maiores *valores estado-ação* passam a ser S_2, S_4, S_5, S_6 e S_{10}, e seleciona-se aleatoriamente o S_6. Após monitorar S_6 no período $t_{6,7}$ e obter a variação correspondente, aplica-se a *regra* associada.

Nesse momento, verifica-se que ocorre a satisfação da regra, o que implica em somar a *recompensa* 2, multiplicada pela taxa de aprendizado α, ao *valor estado-ação*, o qual é multiplicado por um menos a taxa de aprendizado α (0.2) (figura 4.5 (d)). Novamente, a escolha do sucessor acarreta no sorteio entre S_7, S_8 e S_9, uma vez que esses estados possuem o mesmo *valor estado-ação*. O próximo estado selecionado corresponde a S_8. Através da monitoração desse estado, obtêm-se as variações conforme os dados amostrados no intervalo de $t_{8,9}$. Substituindo os `OIDs` presentes na expressão por esses valores, percebe-se que ocorre a satisfação da regra de avaliação. O *valor estado-ação* é, então, recompensado e o próximo passo seria selecionar o nodo sucessor (figura 4.5 (e)). No entanto, como vê-se na árvore de estados da figura 4.4 (a), S_8 é um nodo terminal, o que implica em continuar nesse estado (figura 4.5 (f)) até que não ocorra mais a satisfação dessa regra.

Como o sistema chegou a um dos possíveis estados terminais, tem-se o primeiro aprendizado realizado a partir da monitoração da rede. Dessa forma, o comportamento não esperado da rede é descrito através de uma seqüência de estados, a qual pode ser reutilizada em outras situações similares. A seqüência de estados S_0, S_5 e S_8 indica a presença de uma quantia anormal de descarte de datagramas IP devido à presença de endereços de destino não alcançáveis. A detecção dessa anomalia é uma possível evidência de que a rede pode estar sofrendo algum tipo de intrusão como, por exemplo, inundação de pacotes aleatórios *(TARGA3)*. Outra possibilidade consiste na ocorrência de falha no processo de atualização das rotas de roteamento no dispositivo monitorado. Sem o apoio de um mecanismo adaptativo de monitoração, tal diagnóstico seria dificultado pela ausência de precisão a respeito da situação detectada. Portanto, a principal característica desse modelo proposto consiste no aprendizado contínuo dos comportamentos da rede analisada, por meio de adaptações dinâmicas do conjunto de objetos gerenciáveis a fim de propiciar o acompanhamento e a observação mais precisa.

5 IMPLEMENTAÇÃO

Nesse capítulo é apresentado o sistema que foi desenvolvido a partir do modelo de monitoração adaptativa proposto na seção 4.1 com a finalidade de proporcionar, além do seu funcionamento base, a avaliação da solução como um todo. O capítulo está organizado da seguinte forma. Inicialmente, na seção 5.1, é descrita a arquitetura geral do sistema. Na seção 5.2 são introduzidos os componentes dessa arquitetura, incluindo o detalhamento de suas funções e as interações estabelecidas entre eles. Por fim, tem-se, na seção 5.3, a apresentação das tecnologias envolvidas na elaboração desse sistema.

5.1 Arquitetura

A fim de consolidar o modelo de monitoração adaptativa proposto na seção 4.1, desenvolveu-se um sistema, cuja arquitetura é constituída por dois subsistemas, o elemento autonômico e o subsistema de apoio, conforme ilustrado na figura 5.1. O elemento autonômico confere adaptação ao processo de monitoração a partir do aprendizado obtido por meio da percepção de alterações comportamentais do ambiente e da experimentação de ações sobre ele. Já o subsistema de apoio foi desenvolvido com o intuito de prover suporte à validação do elemento autonômico. A decisão de implementar esse subsistema foi tomada por propiciar uma análise mais exaustiva do modelo, devido à possibilidade de combinar diferentes situações para compor a dinâmica de comportamento do ambiente.

Com relação à arquitetura ilustrada na figura 5.1, o elemento autonômico é composto pelo gerente autonômico e pelo processo de monitoração. O primeiro componente é responsável por determinar o conjunto de variáveis que serão monitoradas, usando como base o modelo descrito na seção 4.1. Já o segundo componente tem por função realizar as consultas SNMP de acordo com o conjunto de variáveis definidas pelo gerente autonômico. Inicialmente, o administrador interage com o gerente autonômico (1) com o objetivo de fornecer a árvore de objetos a serem monitorados, de escolher a política a ser utilizada pelo algoritmo *Q-Learning* e de definir o intervalo de monitoração. Após, tem-se o envio de uma ação (2) disparada pelo gerente autonômico ao processo de monitoração informando os objetos gerenciados e o intervalo de consulta.

Por sua vez, o processo de monitoração dispara requisições SNMP ao subsistema de apoio (3) e decodifica as respectivas respostas (4). Além disso, ele também faz o controle sobre a temporização dessas consultas a partir do intervalo de monitoração provido pela ação (2). Os retornos das requisições SNMP são processados (4) e enviados ao gerente autonômico (5). A partir disso, o gerente tem os subsídios necessários para tomar a decisão em relação à ampliação ou à redução dos objetos de análise do próximo intervalo. De posse da resolução, envia-se uma nova ação (2) ao processo de monitoração e, assim, sucessivamente. Esse ciclo permanece indefinidamente até que ocorra a intervenção do

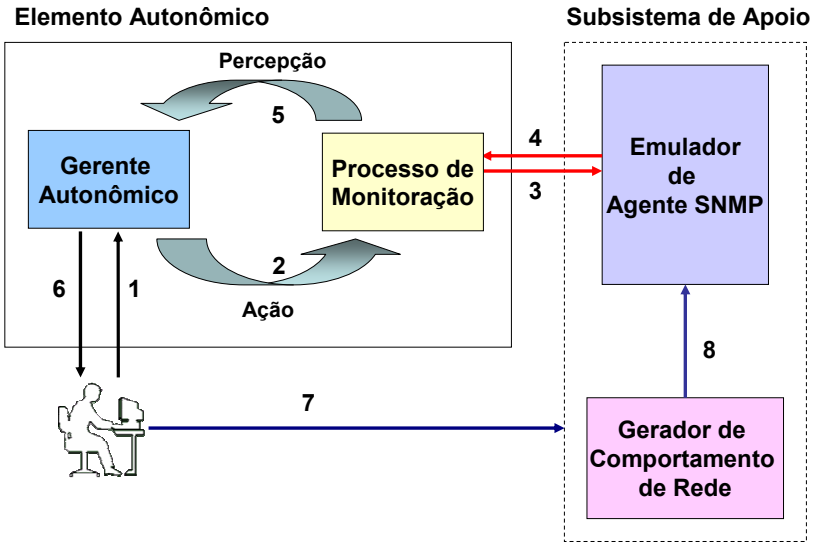


Figura 5.1: Arquitetura para monitoração adaptativa

administrador para finalizar esse procedimento. Como resposta, tem-se a evolução dos objetos gerenciados, que pode ser acompanhado pelo operador, bem como a contabilização dos valores dos objetos retornados em cada ciclo de monitoração (6).

Por fim, tem-se o subsistema de apoio, o qual é composto por um gerador de comportamento de rede e por um emulador de agente SNMP. O primeiro componente é responsável por criar e fornecer situações hipotéticas que poderiam ocorrer em um ambiente de análise real. O administrador interage com esse gerador (7) por meio da construção de perfis de situações de redes que almeja-se replicar ao longo do tempo de monitoração desejado. Nesse trabalho, o emprego da palavra situação deve ser entendido como sinônimo de um comportamento de rede que foge ao padrão de normalidade durante um determinado período de tempo. O emulador de agentes possui a função de responder as requisições SNMP realizadas pelo processo de monitoração (4 e 3). Essa resposta é originada a partir das inúmeras situações previamente criadas (8) pelo gerador de comportamento de rede.

5.2 Componentes da Arquitetura

Essa seção possui o objetivo de apresentar aspectos específicos de implementação dos componentes da arquitetura previamente abordada. Cada componente será detalhado com foco nas suas funcionalidades e nos elementos necessários para prover as interações entre eles.

5.2.1 Gerente Autonômico

O Gerente Autonômico consiste no principal componente do sistema devido ao seu papel estratégico em relação à tomada de decisão das próximas ações a serem executadas em decorrência das percepções obtidas sobre o comportamento da rede. Como visto anteriormente, suas entradas são compostas pela árvore de objetos a serem monitorados, pelo tipo de política a ser utilizada pelo algoritmo Q-Learning (seção 2.2.2) e pelo intervalo de monitoração (figura 5.1, fluxo 1). Todas essas entradas são descritas no formato XML que será apresentado e explicado a seguir. Existe uma quarta entrada, que compreende a alimentação interna do gerente baseada nas percepções obtidas por meio das requisições

SNMP (figura 5.1, fluxo 5).

Inicialmente tem-se a declaração dos parâmetros de configuração do sistema (<config>), os quais compreendem o tipo da política (<policyType>), a taxa de aprendizado (<policyLearningRate>), o tempo de suspensão (<suspenseTime>), a unidade do tempo de suspensão (<suspenseTimeUnit>) o intervalo de tempo de monitoração (<monitoringTimeInterval>) e a sua unidade (<timeUnit>). Os valores aceitáveis para o tipo de política são *Boltzmann*, *Egreedy* e Bloqueio Temporário. No caso dos valores *Boltzmann* ou *Egreedy* serem selecionados, então a próxima informação sobre a taxa de aprendizado é utilizada pelo algoritmo. No entanto, se o valor for igual a Bloqueio Temporário são processadas pelo algoritmo as próximas *tags* de <suspenseTime> e <suspenseTimeUnit>, que correspondem ao tempo que um estado deve ficar suspenso e a sua respectiva unidade que pode variar entre milessegundos e segundos. Os parâmetros restantes independem da política, pois estão associados ao intervalo de tempo de monitoração e sua unidade que pode variar em segundos, minutos e horas.

```
1  <config>
2    <policyType>Boltzmann</policyType>
3    <policyLearningRate>0.1</policyLearningRate>
4    <suspenseTime>30</suspenseTime>
5    <suspenseTimeUnit>ms</suspenseTimeUnit>
6    <monitoringTimeInterval>30</monitoringTimeInterval>
7    <timeUnit>s</timeUnit>
8  </config>
9  <NodeTree>
10   <name>S0</name>
11   <OIDS>
12      <string>.1.3.6.1.2.1.4.3.0</string>
13      <string>.1.3.6.1.2.1.4.10.0</string>
14   </OIDS>
15   <rule>((!=.1.3.6.1.2.1.4.3.0>1000) || (!=.1.3.6.1.2.1.4.10.0>1000))</rule>
16   <value>0.0</value>
17   <reward>2</reward>
18   <penalty>1</penalty>
19   <loopDetected>false</loopDetected>
20   <childNodeTree>
21     <NodeTree>
22         <name>S1</name>
23         <OIDS>
24           <string>.1.3.6.1.2.1.4.3.0</string>
25           <string>.1.3.6.1.2.1.4.4.0</string>
26         </OIDS>
27         <rule>((!=.1.3.6.1.2.1.4.4.0/!=.1.3.6.1.2.1.4.3.0)>= 0,3)</rule>
28         <value>0.0</value>
29         <reward>2</reward>
30         <penalty>1</penalty>
31         <loopDetected>false</loopDetected>
32         <parentNodeTree reference="../../.." />
33         <childNodeTree />
34         <suspenseTime>0</suspenseTime>
35     </NodeTree>
36     <NodeTree>
37         ...
38      </NodeTree>
39     ...
40   </childNodeTree>
41 </NodeTree>
```

Figura 5.2: Exemplo de arquivo de configuração inicial do sistema.

O próximo conjunto de dados da figura 5.2 corresponde à árvore completa dos objetos de monitoração. A *tag* presente na linha 9 desse arquivo (<NodeTree>) repre-

senta o nó raiz do sistema. A estrutura desse nó é formada pelo: *(i)* nome do estado (<name>); *(ii)* pela definição dos OIDs que compõe o estado (<OIDS>); *(iii)* pela regra de avaliação usada para promover a transição entre estados (<rule>); *(iv)* pelo valor estado-ação (<value>); *(v)* pela recompensa (<reward>); *(vi)* pela penalidade (<penalty>); *(vii)* pela detecção de *loop* (<loopDetected>); *(viii)* e, por fim, pelos nós filhos (<childNodeTree>). Os atributos valor estado-ação e detecção de *loop* são de uso exclusivo do sistema e, originalmente, não devem ser preenchidos com valores diferentes do padrão 0.0 e *false*, respectivamente.

A descrição dos nós filhos repete a mesma estrutura anteriormente apresentada. Contudo, encontram-se dois elementos adicionais: a referência para o nó pai (<parentNode Tree reference="../../.."/>) e o tempo de suspensão (<suspenseTime>). Esse atributo de tempo de suspensão possui o seu uso restrito ao algoritmo de Bloqueio Temporário e também recomenda-se utilizar o valor padrão inicial igual a zero. Os nós filhos se diferenciam dos demais por estarem localizados abaixo da *tag* <childNodeTre e>, conforme mostrado na linha 20. Não existe uma quantia limite de nós filhos que um nó pai pode possuir. No entanto, deve-se ter o bom senso para elaborar uma distribuição equilibrada de estados na árvore a fim de não comprometer o desempenho do sistema.

Depois de realizar a leitura da árvore de estados, o gerente autonômico cria uma lista de nós duplamente encadeados a fim de permitir a navegação tanto em direção à raiz quanto no sentindo dos nós folhas. De posse das entradas, o gerente autonômico dispara a primeira ação que corresponde à coleta dos valores das variáveis do nó raiz realizada pelo processo de monitoração, o qual será explicado na seqüência. Os dados obtidos são processados e enviados ao gerente autonômico que aplica a regra para obter a direção da transição a ser realizada. No caso de avançar em direção aos estados filhos, o gerente emprega a política previamente estabelecida pelo operador (Egreedy, *Boltzmann* ou Bloqueio Temporário) com o objetivo de escolher o próximo estado. Assim, tem-se a definição de uma nova ação e, dessa forma, inicia-se um ciclo até o término do período estabelecido para monitoração. Maiores informações sobre o algoritmo usado por esse componente podem ser encontrados na seção 4.1.

Como visto anteriormente, tem-se, como uma das saídas desse componente, a ação de monitoração (figura. 5.1, fluxo 2), a qual é utilizada pelo processo de monitoração detalhado na sub-seção seguinte. Essa ação corresponde ao envio de um objeto ao processo de monitoração, que é composto pelo conjunto de OIDs a ser usado na requisição SNMP e pelo intervalo de monitoração. A fim de obter uma análise precisa da eficácia dos algoritmos utilizados pelo gerente autonômico foi gerada uma saída de estatística, ilustrado na figura 5.3. Nela são descritos o tipo da política (linha 2), a taxa de aprendizado (linha 3), os valores parciais por cada situação anormal percebida pelo gerente (por exemplo, <S1>0</S1>) e, por último, tem-se a soma total desses comportamentos (linha 11). Para o cálculo dessa estatística, é necessário conhecer a quantidade de situações criadas pelo gerador de comportamento de rede que será apresentado na seqüência.

Como última saída, tem-se o *log* dos estados visitados pelo algoritmo de aprendizado, ilustrado nas figuras 5.4 e 5.5. Ambas as figuras possuem o mesmo conjunto de dados, porém optou-se por dispor os dados em forma de tabela (figura 5.5) para facilitar a visualização e, assim, propiciar melhor compreensão do arquivo de *log* em XML nesse trabalho. Essa tabela apresenta um contador de interações, o estado atual, o valor-estado ação e ação escolhida, que corresponde ao próximo estado a ser visitado. No exemplo, tem-se as 20 primeiras iterações de monitoração, na qual se detectou a situação $S8$ no passo 8. Antes disso, os estados folhas visitados foram $S2$, $S5$ e $S7$. Porém, não houve satisfação

das condições descritas nas regras associadas a cada um desses nós, uma vez que não há permanência consecutiva de cada estado na coluna que indica o próximo estado a ser visitado.

```
<result>
  <policyType>Boltzmann</policyType>
  <policyLearningRate>0.1</policyLearningRate>
  <S1>54</S1>
  <S2>36</S2>
  <S4>29</S4>
  <S5>46</S5>
  <S7>107</S7>
  <S8>98</S8>
  <S9>98</S9>
  <S11>67</S11>
  <S12>65</S12>
  <sum>600</sum>
</result>
```

Figura 5.3: Exemplo de valores parciais para cada situação detectada e valor total das situações detectadas, utilizando a política Boltzmann (α = 0.1).

5.2.2 Processo de Monitoração

O processo de monitoração é responsável por invocar as primitivas de construção, de envio e de recepção dos pacotes SNMP por meio de uma API. Ele foi desenvolvido como um elemento interno ao sistema, o que possibilita a comunicação interprocessos dele com o gerente autonômico. Como entrada, tem-se um conjunto de `OIDs` do nó a ser monitorado pelo gerente autonômico e o intervalo de tempo a ser utilizado entre as consultas (figura 5.1, fluxo 2).

No processo de monitoração tem-se a chamada para a rotina responsável pela construção do pacote SNMP do tipo *Get-Request*, o qual utiliza o protocolo de transporte UDP. Ele contém todos os `OIDs` obtidos a partir do objeto enviado pelo gerente autonômico. Esse pacote é enviado para o emulador de agente SNMP (figura 5.1, fluxo 3), que será apresentado na seqüência. Esse envio é realizado em dois momentos: no início e no final do intervalo de monitoração. Esse componente implementa um *listener*, cuja função consiste em esperar as respostas do emulador de agente. A partir desse processo, obtém-se pacotes SNMP do tipo *Get-Response* (figura 5.1, fluxo 4) contendo os valores obtidos do agente simulado e envia-se esse dados para o gerente autonômico por meio de mecanismos de comunicação interprocessos (figura 5.1, fluxo 5). Conforme o tipo de regra, tem-se o processamento dos dados como, por exemplo, a diferença entre as duas aferições realizadas no intervalo de monitoração. No caso da regra considerar somente um valor, este sempre corresponderá ao dado da última requisição.

Para contabilizar o tempo de espera entre as duas requisições realizadas, o processo de monitoração implementa um temporizador que é disparado a partir da primeira requisição SNMP. Após atingir o intervalo de tempo definido para esse conjunto de monitoração ele faz a segunda consulta. No entanto, existe um problema conhecido a respeito da latência de rede como fator de influência no processo de monitoração a ser explicado na seqüência.

A figura 5.6 ilustra duas linhas do tempo a respeito da influência do ambiente de rede no processo de monitoração e quando essa é desconsiderada pela utilização de simulação local. Para ambas linhas de tempo, foram adotados os mesmos intervalos de monitoração α_1 e α_2, os quais são fornecidos pelo gerente autonômico ao processo de monitoração.

```
<log>
    <counter>0</counter>
    <state>S0</state>
    <Value_Action_State>0.0</Value_Action_State>
    <next_state>S2</next_state>
    <counter>1</counter>
    <state>S2</state>
    <Value_Action_State>0.0</Value_Action_State>
    <next_state>S0</next_state>
    <counter>2</counter>
    <state>S0</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S5</next_state>
    <counter>3</counter>
    <state>S5</state>
    <Value_Action_State>0.0</Value_Action_State>
    <next_state>S0</next_state>
    <counter>4</counter>
    <state>S0</state>
    <Value_Action_State>0.38</Value_Action_State>
    <next_state>S6</next_state>
    <counter>5</counter>
    <state>S6</state>
    <Value_Action_State>0.0</Value_Action_State>
    <next_state>S7</next_state>
    <counter>6</counter>
    <state>S7</state>
    <Value_Action_State>0.0</Value_Action_State>
    <next_state>S6</next_state>
    <counter>7</counter>
    <state>S6</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S8</next_state>
    <counter>8</counter>
    <state>S8</state>
    <Value_Action_State>0.0</Value_Action_State>
    <next_state>S8</next_state>
    <counter>9</counter>
    <state>S8</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S8</next_state>
    <counter>10</counter>
    <state>S8</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S8</next_state>
    <counter>11</counter>
    <state>S8</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S8</next_state>
    <counter>12</counter>
    <state>S8</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S8</next_state>
    <counter>13</counter>
    <state>S8</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S8</next_state>
    <counter>14</counter>
    <state>S8</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S8</next_state>
    <counter>15</counter>
    <state>S8</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S8</next_state>
    <counter>16</counter>
    <state>S8</state>
    <Value_Action_State>0.2</Value_Action_State>
    <next_state>S6</next_state>
    <counter>17</counter>
    <state>S6</state>
    <Value_Action_State>0.38</Value_Action_State>
    <next_state>S0</next_state>
    <counter>18</counter>
    <state>S0</state>
    <Value_Action_State>0.542</Value_Action_State>
    <next_state>S0</next_state>
    <counter>19</counter>
    <state>S0</state>
    <Value_Action_State>0.3878000</Value_Action_State>
    <next_state>S0</next_state>
    <counter>20</counter>
    <state>S0</state>
    <Value_Action_State>0.3878000</Value_Action_State>
    <next_state>S0</next_state>
</log>
```

Figura 5.4: Estrutura do *log* exemplo em XML

Counter	State	Value_Action_State	Next_State
0	S0	0.0	S2
1	S2	0.0	S0
2	S0	0.2	S5
3	S5	0.0	S0
4	S0	0.38	S6
5	S6	0.0	S7
6	S7	0.0	S6
7	S6	0.2	S8
8	S8	0.0	S8
9	S8	0.2	S8
10	S8	0.2	S8
11	S8	0.2	S8
12	S8	0.2	S8
13	S8	0.2	S8
14	S8	0.2	S8
15	S8	0.2	S8
16	S8	0.2	S6
17	S6	0.38	S0
18	S0	0.542	S0
19	S0	0.3878000	S0
20	S0	0.3878000	S0

Figura 5.5: Visualização do *log* exemplo em formato de tabela

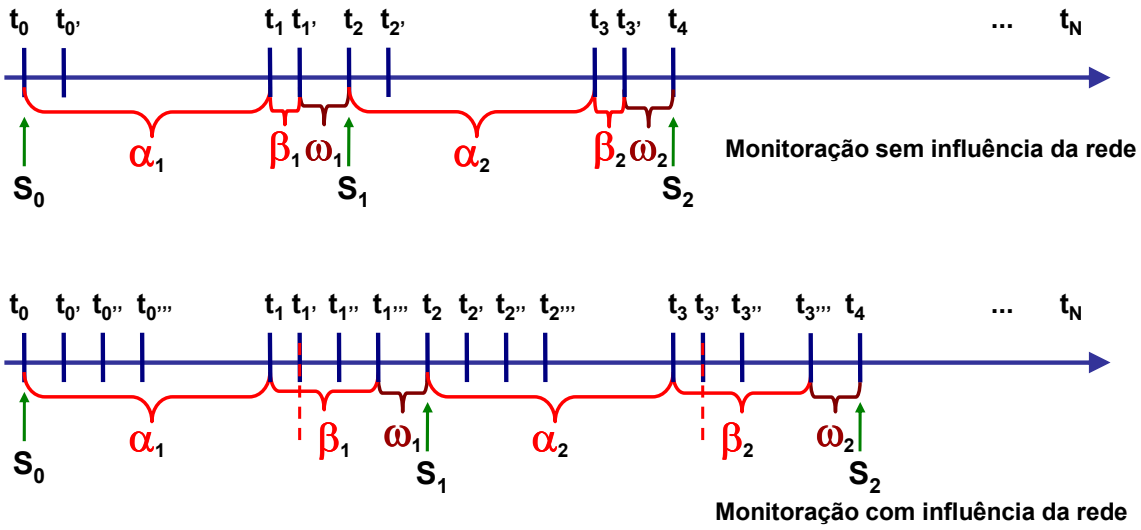


Figura 5.6: Problema da precisão no intervalo de monitoração

Na primeira linha do tempo foi desconsiderada a latência de rede no envio e na resposta da requisição SNMP. Dessa forma, almeja-se desprezar os períodos correspondentes aos atrasos de rede no processo de monitoração. Esse atraso pode ser ocasionado por eventuais congestionamentos, pelo baixo poder de processamento dos dispositivos gerenciáveis, pela longa distância física, por erros de configuração dos equipamentos de rede, entre outras causas. Essas lacunas de tempo geram descontinuidade no espaço amostral de monitoração.

Observe, na primeira linha, a presença de $t_x - t_{x'}$, a qual corresponde ao tempo de processamento do dispositivo gerenciado. O processamento no dispositivo envolve as tarefas de recepção de consultas SNMP, de coleta de dados e de envio de respostas SNMP à aplicação gerente. O tempo gasto nessas tarefas, no caso da segunda amostra de cada período, implica na existência de lacuna de tempo (β_x) no processo de monitoração, apesar desta ficar restrita à ordem de grandeza de milissegundos. Assim, a desconsideração dessa lacuna de tempo não ocasiona nenhum impacto no processo de amostragem. Existem também lacunas de tempo representadas por ω_x, as quais consistem no tempo de processamento despendido na avaliação do estado a partir dos dados coletados até determinação do próximo estado S_x, ao qual o sistema deverá evoluir.

Já na segunda linha de tempo, tem-se a presença da latência de rede, representada pelas lacunas de tempo $t_x - t_{x'}$ no envio da requisição SNMP e $t_{x''} - t_{x'''}$ na resposta da consulta SNMP. Observe que essas lacunas de tempos podem ser significativas (maiores que 1s), acarretando em perdas expressivas de informações de monitoração, e variáveis, produzindo diferentes atrasos β_1 e β_2 suscetíveis às condições da rede. Assim, é difícil estipular ou prever a quantia exata de tempo envolvida na latência de rede para que se pudesse adicionar um coeficiente e, dessa maneira, re-adequar o intervalo de monitoração a cada requisição SNMP. A re-adequação do intervalo de monitoração faz-se necessária para garantir a precisão dos instantes de amostragem e evitar a perda de dados sensíveis à monitoração. Cada β_x ainda é composto pelo tempo de processamento no dispositivo gerenciado ($t_{x'} - t_{x''}$).

5.2.3 Gerador de Comportamento da Rede

O gerador de comportamento da rede é responsável por criar arquivos no formato XML, os quais correspondem ao fluxo de comportamentos em um dado intervalo. Esse fluxo é criado a partir das informações disponibilizadas pelo operador (figura 5.1, fluxo 7), as quais compreendem: os diferentes perfis, a quantidade de horas de simulação da rede e o intervalo de monitoração. Cada perfil representa um comportamento de rede que se deseja replicar durante um período de tempo. Observe a figura 5.7, na qual estão descritos diferentes comportamentos pela *tag* `<behavior>`. Cada um deles apresenta um conjunto de dados constituídos por: (*i*) nome que caracteriza o comportamento (`<name>`); (*ii*) descrição ou anotação sobre ele (`<description>`); (*iii*) tempo de duração desse comportamento expresso em unidades de monitoração (`<timeDuration>`); (*iv*) `OIDs` das variáveis (`<OIDs>`); (*vi*) e, por fim, seus valores característicos (`<value>`). A partir dessas informações é gerada uma quantidade de arquivos suficiente para compreender todo esse período de análise conforme expresso pelo algoritmo 3.

Algoritmo 3 Gerador de Comportamento

```
1: GeraComportamento(ListaSituação ls, int intervaloMonitoração,Situação situação-
   Normal){
2:
3: Lista listaSituação;
4: Enquanto (intervaloMonitoração > 0) faça
5:    Se (listaSituação == Lista_Vazia) então {
6:       listaSituação = ls;
7:    }
8:    //Escolhe aleatoriamente uma posição da lista de situações
9:    posição = Obtém_Inteiro(ls.tamanho());
10:
11:   situação = listaSituação.remove(pos);
12:
13:   Para (i = 0 até UMComportamento[pos])
14:      CriaFluxoSituação(situação,i);
15:   fim Para
16:
17:   intervaloMonitoração = intervaloMonitoração - UMComportamento[pos];
18:
19:   //Escolhe aleatoriamente uma quantia de unidades de monitoração
20:   //para o perfil de normalidade de comportamento de rede
21:   qtd = Obtém_Inteiro(UMNormal);
22:
23:   Para (i = 0 até qtd)
24:      CriaFluxoSituação(situaçãoNormal,i);
25:   fim Para
26:
27:   intervaloMonitoração = intervaloMonitoração - qtd;
28:
29: fim Enquanto
```

O gerador faz, de maneira aleatória, as intercalações dos diferentes perfis de compor-

```
<behaviors>
    <behavior>
        <name>Situação 0</name>
        <description>Comportamento a ser simulado de normalidade</description>
        <timeDuration>32</timeDuration>
        <variables>
            <OIDs>
                <string>.1.3.6.1.2.1.4.3.0</string>
                <string>.1.3.6.1.2.1.4.4.0</string>
                <string>.1.3.6.1.2.1.4.3.0</string>
                <string>.1.3.6.1.2.1.4.5.0</string>
                <string>.1.3.6.1.2.1.4.10.0</string>
                <string>.1.3.6.1.2.1.4.14.0</string>
                ...
            </OIDs>
            <value>10</value>
            <value>10</value>
            <value>10</value>
            <value>10</value>
            <value>10</value>
            <value>10</value>
            ...
        </variables>
    </behavior>
    <behavior>
        <name>Situação 1</name>
        <description>Comportamento a ser simulado número 1</description>
        <timeDuration>20</timeDuration>
        <variables>
            <OIDs>
                <string>.1.3.6.1.2.1.4.3.0</string>
                <string>.1.3.6.1.2.1.4.4.0</string>
                <string>.1.3.6.1.2.1.4.10.0</string>
            </OIDs>
            <value>17</value>
            <value>17</value>
            <value>17</value>
        </variables>
    </behavior>
    <behavior>
        <name>Situação 2</name>
        <description>Comportamento a ser simulado número 2</description>
        <timeDuration>16</timeDuration>
        <variables>
            <OIDs>
                <string>.1.3.6.1.2.1.4.3.0</string>
                <string>.1.3.6.1.2.1.4.5.0</string>
                <string>.1.3.6.1.2.1.4.10.0</string>
            </OIDs>
            <value>18</value>
            <value>18</value>
            <value>18</value>
        </variables>
    </behavior>
    ....
</behaviors>
```

Figura 5.7: Exemplo da descrição dos perfis de comportamento de redes

tamento simulados. Para tornar mais equilibrada a diversidade de situações geradas, foi adotado que todos os perfis pré-estabelecidos pelo operador são armazenados em uma lista (linha 6) e, uma vez escolhido aleatoriamente um perfil (linha 9), ele é removido (linha 11). Esse procedimento evita seleções sucessivas de um mesmo perfil de comportamento, o que assegura a mesma probabilidade de escolha para todos.

De posse do perfil escolhido, o conjunto de valores característicos é replicado (linha 14) para compor o fluxo de monitoração de acordo com a quantidade pré-determinada de unidades de monitoração representada pela variável *UMComportamento* (linha 13). O valor dessa variável é populado conforme o valor descrito pela *tag* `<timeDuration>` (figura 5.7). Assim, cada comportamento pode apresentar distintas unidades de monitoração. Observe, ainda, que decrementa-se as unidades de monitoração do valor correspondente ao período de monitoração (linha 17).

O próximo passo desse algoritmo consiste na possível inclusão de um perfil que representa a situação de normalidade da rede (linha 24). A escolha da quantidade de unidades de monitoração é realizada de maneira aleatória e pode-se selecionar uma quantia igual a 0 (linha 21). Ou seja, o perfil de normalidade pode não ser incluso e, assim, pode-se ter duas situações anormais consecutivas no fluxo de monitoração gerado. Como anteriormente visto, a quantia representada por *UMNormal* (linha 21) no algoritmo é obtida a partir da *tag* `<timeDuration>` (figura 5.7). Por fim, decrementa-se a quantia de unidades de monitoração correspondente ao comportamento de normalidade (linha 27) do período de monitoração.

Esse processo repete-se até que seja criado o fluxo correspondente ao período de monitoração almejado pelo operador. Como saída desse gerador de situações (figura. 5.1, fluxo 8), tem-se um conjunto de arquivos XML que possuem o mesmo padrão mostrado na figura 5.8. A *tag* `<variable>` corresponde ao `OID` a ser monitorado enquanto que a *tag* `<value>` possui dois valores. O primeiro valor corresponde ao instante inicial de monitoração enquanto que o segundo consiste no instante final do intervalo. Tem-se ainda uma outra saída que descreve para cada comportamento presente no arquivo de perfil (figura 5.7) a quantidade de situações geradas e a soma total delas.

```
1  <MonitoredData>
2    <variable>.1.3.6.1.2.1.4.3.0</variable>
3    <value>
4      <float>0.0</float>
5      <float>10.0</float>
6    </value>
7  </MonitoredData>
8
9  <MonitoredData>
10   <variable>.1.3.6.1.2.1.4.10.0</variable>
11   <value>
12     <float>0.0</float>
13     <float>10.0</float>
14   </value>
15 </MonitoredData>
```

Figura 5.8: Exemplo da representação dos valores dos objetos gerenciáveis utilizados pelo processo de monitoração

5.2.4 Emulador de Agente

O Emulador de Agente possui a função de recepção (figura 5.1, fluxo 3), processamento e envio de pacotes SNMP (figura 5.1, fluxo 4) a partir dos dados disponibilizados

pelo gerador de comportamento de rede. Esse componente implementa um *listener* que fica esperando por pacotes SNMP na porta 161. Após a recepção do pacote, tem-se a extração dos `OIDs` a fim de utilizá-los na obtenção dos valores presentes nos arquivos XML gerados (figura 5.1, fluxo 8) pelo componente anteriormente apresentado. Esse emulador apresenta ainda um acumulador usado na identificação do arquivo XML que corresponde ao instante de monitoração analisado.

Após obter os valores, esses são enviados para o processo de monitoração por meio de uma API. Uma característica desse emulador de agente corresponde à capacidade de prover suporte a qualquer tipo de MIB, sendo elas proprietárias ou não. A decisão de implementar um emulador de agente justifica-se pelo ganho obtido por meio do controle sobre os comportamentos a serem simulados na rede. Assim, é possível obter as informações necessárias para avaliar com precisão o desempenho de sistema.

5.3 Tecnologias Utilizadas

No desenvolvimento de todos os componentes da arquitetura apresentada anteriormente foi utilizada a linguagem Java (JAVA, 2008). A sua escolha baseou-se na propriedade de portabilidade dessa plataforma de modo a permitir a execução das aplicações desenvolvidas nos diferentes sistemas operacionais existentes. Para executar essas aplicações é necessário ter a *Java Virtual Machine* - JVM instalada. O ambiente de desenvolvimento usado na implementação dos componentes do sistema foi o *Netbeans* (NETBEANS, 2008).

O gerente autonômico, o emulador de agentes e o gerador de comportamento de rede necessitam da biblioteca *XStream* (XSTREAM, 2008), cuja funcionalidade permite realizar a leitura de arquivos no formato XML (*Extensible Markup Language*). Essa biblioteca também permite a escrita de dados no padrão XML, o que torna possível o armazenamento da árvore treinada, contendo os valores estado-ação aprendidos. A escolha da representação da árvore de estados no formato XML visa facilitar a sua declaração, bem como tornar rápido o procedimento de conversão dessa estrutura para objetos persistidos em memória. Para a elaboração do *log* de visitação dos estados foram utilizadas as bibliotecas *XStream* e *SuperCVS* (SUPERCSV, 2008). Essa última biblioteca permite o armazenamento dos dados no formato CSV (*Comma Separated Values*), o que facilita a interpretação visual a partir dos dados dispostos no formato de tabela. Outra consideração a ser feita sobre o gerente autonômico consiste na implementação do algoritmo *Q-Learning* e das políticas ϵ-Gulosa, *Boltzmann* e Bloqueio Temporário em Java, as quais são contribuições desse trabalho.

O processo de monitoração e o emulador de agentes usam a API *SNMP4J* (SNMP4J, 2008) para a criação, envio e recepção de pacotes SNMP. Além dessas opções, essa API permite o envio de mensagens baseadas nos eventos, ou seja, *traps* SNMP, porém essa funcionalidade não é suportada nessa versão do sistema.

6 AVALIAÇÃO EXPERIMENTAL

Este capítulo apresenta uma avaliação parcial do modelo, com base em um estudo de caso, realizada a partir da implementação da arquitetura para monitoração adaptativa, descrita anteriormente na seção 5.1. Na seção 6.1, apresenta-se a metodologia e o cenário utilizados na avaliação experimental. Em seguida, na seção 6.2, averigua-se a eficácia das políticas na detecção de situações produzidas pelo gerador de comportamento. Na seção 6.3, utiliza-se o cenário treinado do experimento anterior para avaliar questões relacionadas à eficiência das políticas na detecção de comportamentos que não sejam inéditos para a árvore treinada.

6.1 Caracterização da Metodologia e do Cenário Empregado

A avaliação experimental foi realizada sobre um ambiente de rede simulado, intitulado sistema de apoio, apresentado no capítulo 5, ao invés de empregar observações de uma rede real. Dessa maneira, pode-se combinar diferentes situações para compor dinâmicas de comportamento do ambiente desejadas, o que, comparando à rede real, despenderia de um grande período de monitoração e, mesmo assim, não há garantia da observação de todas elas. Nesse trabalho, o emprego da palavra *situação* deve ser entendido como sinônimo de um comportamento de rede que foge ao padrão de normalidade durante um determinado período de tempo. Já *não estacionariedade* refere-se ao grau com que situações distintas podem ser observadas. Por exemplo, um ambiente de rede com grau de não estacionariedade igual a 5 significa que existem cinco situações que se alternam entre si e com o padrão de normalidade.

Nessa perspectiva, o primeiro objetivo da avaliação foi determinar a taxa de detecção de situações em ambientes com distintos graus de não estacionariedade, usando como base o modelo e o estudo de caso anteriormente apresentados. O segundo objetivo, por sua vez, consistiu na avaliação do desempenho do elemento autonômico a partir do reuso do conhecimento aprendido sob o estudo de caso. Nesse caso, verificou-se a quantidade de iterações necessárias pelo elemento autonômico para detectar uma situação a partir da árvore de estados previamente treinada para esses comportamentos.

Para analisar o quão eficaz é o modelo de monitoração adaptativa proposto, definiu-se uma árvore de estados inspirada a partir de um cenário de gerência de redes conhecido, como encontrado em Nunes (NUNES, 1997) e em Stallings (STALLINGS, 1997). Para cada estado, assumiu-se que os valores ideais para a caracterização do comportamento anormal em razão das variáveis que compõem a regra de avaliação, considerando esse estudo de caso, consiste em 30% ou superior. Essa árvore de estados corresponde a um dos dados de entrada utilizados pelo elemento autonômico. Conforme pode ser observado na figura 6.1, a árvore de estados é composta por 13 estados, sendo inicialmente populada

com os valores descritos na tabela. Essa tabela lista o estado, os *OIDs*, a *regra de avaliação*, a *recompensa*, a *penalidade*, o *valor estado-ação*, a *detecção de loop* e o *tempo de suspensão* associados a cada estado. Observe que tem-se valores padrões para os atributos *valor estado-ação* (0.0), *detecção de loop* ($false$) e *tempo de suspensão* (0.0). Os valores de *recompensa* e *penalidade* foram mantidos iguais a 2 e -1, respectivamente, para todos os estados do sistema. A análise da variação de seus valores por estado não será objeto de estudo desse trabalho. Os relacionamentos entre os estados (pai-filho) são descritos por meio da árvore presente na figura 6.1.

De posse da árvore de estados anteriormente apresentada, a próxima etapa consiste na criação dos conjuntos de situações pelo gerador de comportamento da rede segundo a metodologia previamente descrita na seção 5.2.3. O gerador de comportamento criou 9 conjuntos de dados, nos quais tem-se o grau de não estacionariedade variando de uma a nove situações. Cada situação possui duração (*timeDuration*) igual a 16, definida em função do número de visitas necessárias aos estados, considerando o pior caso de busca para adequar o conjunto de *OIDs* na tentativa de explicar determinado comportamento.

O pior caso consiste em percorrer todos os estados pertencentes ao mesmo nível e avançar pelo ramo que determina a altura da árvore, repetindo esse mesmo procedimento até o nó folha. O conjunto de dados criado (*OIDs versus* valor), correspondente a uma situação, foi intercalado com o fluxo que representa o padrão de normalidade de comportamento de rede ou com o fluxo que retrata outra situação. Esse padrão de normalidade consistiu na criação de um fluxo de comportamento descrito por variáveis que contém valores aceitáveis dentro do limite de normalidade.

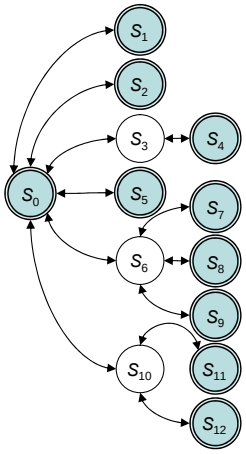
A definição da quantidade de unidades de monitoração desse fluxo é baseada na escolha aleatória, vista na seção 5.2.3. Nesse experimento, usou-se como valor máximo 32 minutos, que equivale ao dobro da busca em profundidade do pior caso anteriormente mencionado. Um exemplo de uma possível intercalação de situações com comportamento normal pode ser visualizada na figura 6.1. Nela, ilustram-se as amostras equivalentes ao período de análise de um mês de monitoração com requisições realizadas no intervalo de um minuto.

6.2 Eficácia no Processo de Detecção de Situações

Com base no cenário exposto acima, realizou-se o primeiro conjunto de experimentos considerando o algoritmo *Q-Learning* dadas as três políticas ϵ-Gulosa, *Boltzmann* e Bloqueio Temporário. Para todas essas políticas utilizou-se a taxa de aprendizado α igual a 0.1 e o fator de desconto γ igual a 0, a fim de desconsiderar o aprendizado dos estados futuros no processo de tomada de decisão da próxima ação.

As duas primeiras políticas (ϵ-Gulosa e *Boltzmann*) possuem, ainda, outro critério de variabilidade, referente à taxa de exploração e ao parâmetro de temperatura, respectivamente. Como esses critérios possuem o mesmo objetivo de influenciar a probabilidade de escolha, o que pode aumentar ou reduzir o grau de exploração entre os estados, decidiu-se, nesse trabalho, referenciá-los como fator de influência. Nos experimentos realizados, considerou-se os valores de 0.1, 0.5 e 0.9 para esses fatores.

Ao invés do fator de influência, a proposta do método guloso considerando Bloqueio Temporário fundamenta-se na suspensão de um estado durante um período determinado. Esse período corresponde ao tempo de processamento da avaliação da regra somado ao tempo transcorrido no processo de aquisição dos objetos monitorados multiplicado pela quantidade de estados presentes no mesmo nível da árvore. Nesse experimento, esse



Estado	OID	Regra	*R.[a]	*P.[b]	*V.[c]	*L.[d]	*T.[e]
S_0	.1.3.6.1.2.1.4.3.0 .1.3.6.1.2.1.4.10.0	$(\Delta.1.3.6.1.2.1.4.3.0 > 1000) \Vert (\Delta.1.3.6.1.2.1.4.10.0 > 1000)$	2	-1	0.0	false	0
S_1	.1.3.6.1.2.1.4.4.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.4.4.0}{\Delta.1.3.6.1.2.1.4.3.0}\right) > 30\%$	2	-1	0.0	false	0
S_2	.1.3.6.1.2.1.4.5.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.4.5.0}{\Delta.1.3.6.1.2.1.4.3.0}\right) > 30\%$	2	-1	0.0	false	0
S_3	.1.3.6.1.2.1.4.14.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.4.14.0}{\Delta.1.3.6.1.2.1.4.3.0}\right) > 30\%$	2	-1	0.0	false	0
S_4	.1.3.6.1.2.1.4.16.0 .1.3.6.1.2.1.4.14.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.4.16.0}{\Delta.1.3.6.1.2.1.4.14.0}\right) > 30\%$	2	-1	0.0	false	0
S_5	.1.3.6.1.2.1.4.8.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.4.8.0}{\Delta.1.3.6.1.2.1.4.3.0}\right) > 30\%$	2	-1	0.0	false	0
S_6	.1.3.6.1.2.1.4.1.0 .1.3.6.1.2.1.4.10.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.4.1.0}{\Delta.1.3.6.1.2.1.4.3.0}\right) > 30\%$ && $(\Delta.1.3.6.1.2.1.4.10.0 > 1000)$	2	-1	0.0	false	0
S_7	.1.3.6.1.2.1.4.11.0 .1.3.6.1.2.1.4.1.0 .1.3.6.1.2.1.4.10.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.4.11.0}{\Delta.1.3.6.1.2.1.4.1.0+\Delta.1.3.6.1.2.1.4.10.0}\right) > 30\%$	2	-1	0.0	false	0
S_8	.1.3.6.1.2.1.4.12.0 .1.3.6.1.2.1.4.1.0 .1.3.6.1.2.1.4.10.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.4.12.0}{\Delta.1.3.6.1.2.1.4.1.0+\Delta.1.3.6.1.2.1.4.10.0}\right) > 30\%$	2	-1	0.0	false	0
S_9	.1.3.6.1.2.1.4.18.0 .1.3.6.1.2.1.4.1.0 .1.3.6.1.2.1.4.10.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.4.18.0}{\Delta.1.3.6.1.2.1.4.1.0+\Delta.1.3.6.1.2.1.4.10.0}\right) > 30\%$	2	-1	0.0	false	0
S_{10}	.1.3.6.1.2.1.5.1.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.5.1.0}{\Delta.1.3.6.1.2.1.4.3.0}\right) > 30\%$	2	-1	0.0	false	0
S_{11}	.1.3.6.1.2.1.5.2.0 .1.3.6.1.2.1.5.1.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.5.2.0}{\Delta.1.3.6.1.2.1.5.1.0}\right) > 30\%$	2	-1	0.0	false	0
S_{12}	.1.3.6.1.2.1.5.9.0 .1.3.6.1.2.1.5.1.0 .1.3.6.1.2.1.5.21.0 .1.3.6.1.2.1.4.3.0	$\left(\frac{\Delta.1.3.6.1.2.1.5.9.0}{\Delta.1.3.6.1.2.1.5.1.0+\Delta.1.3.6.1.2.1.5.21.0}\right) > 30\%$	2	-1	0.0	false	0

[a] Recompensa
[b] Penalidade
[c] Valor estado-ação
[d] Detecção de Loop
[e] Tempo de Suspensão do Estado

Figura 6.1: Árvore de estados e tabela representando uma das entradas do elemento autonômico.

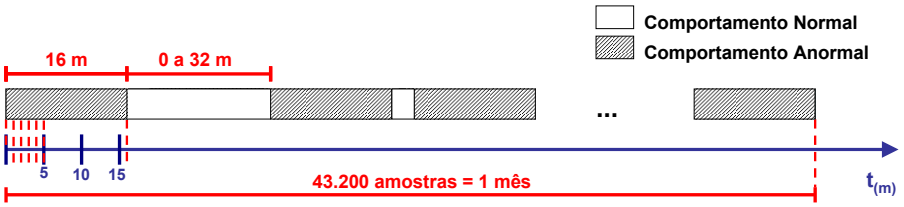


Figura 6.2: Intercalação de valores dos objetos gerenciáveis normais e anormais que compreendem um fluxo de tráfego ou conjunto de dados.

tempo foi medido considerando a regra que apresentou o maior tempo de processamento, ficando restrito na ordem de milisegundos (300 ms). O tempo de aquisição de objetos foi desconsiderado, pois as consultas foram realizadas diretamente ao emulador de agente. Para cada política, dada a taxa de aprendizado igual a 0.1 e os três valores 0.1, 0.5 e 0.9 para o fator de influência, realizou-se experimentos com 5 repetições sobre os conjuntos de dados gerados. Esse baixo número de repetições deve-se a grande quantia de tempo despendida para a simulação equivalente a um mês de monitoração de uma rede, contendo requisições SNMPs a cada 30 segundos e, ainda, considerando do 1° ao 9° grau de não estacionariedade.

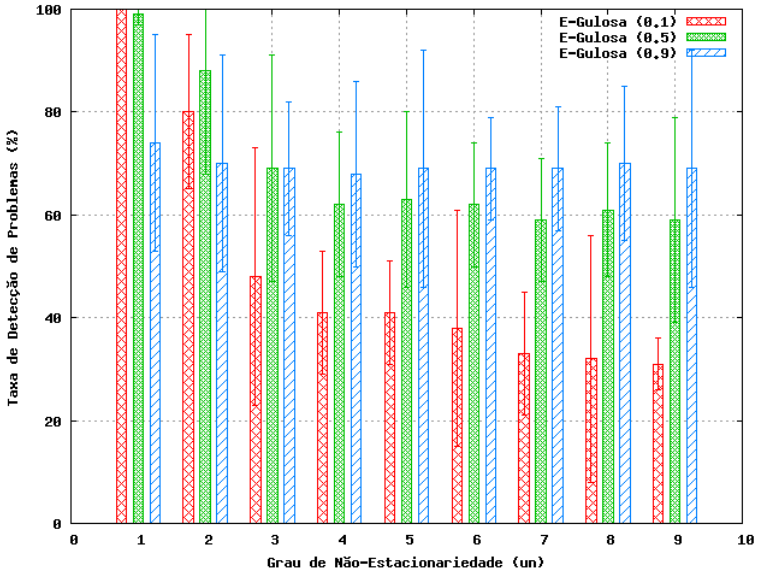
Com relação à taxa de aprendizado, decidiu-se não variá-la inicialmente, pois não se obteria um ganho expressivo de desempenho uma vez que as recompensas e as penalidades possuem o mesmo valor em todos os estados. Dessa forma, a variação da taxa de aprendizado apenas influencia na velocidade da aprendizagem, o que para cenários dinâmicos é melhor manter baixo devido ao constante re-aprendizado.

Os resultados obtidos são ilustrados na figura 6.3. Como pode ser visto em 6.3(a) e 6.3(b), são comparados entre si o mesmo tipo de política contendo diferentes valores para o fator de influência. Nesses gráficos pode-se constatar que a variação do fator de influência implica em resultados distintos, nos quais os melhores podem ser obtidos quando esse fator possui valor igual a 0.9 em ambas as políticas, considerando ambientes cuja variação de não estacionariedade é equivalente ou superior a 5.

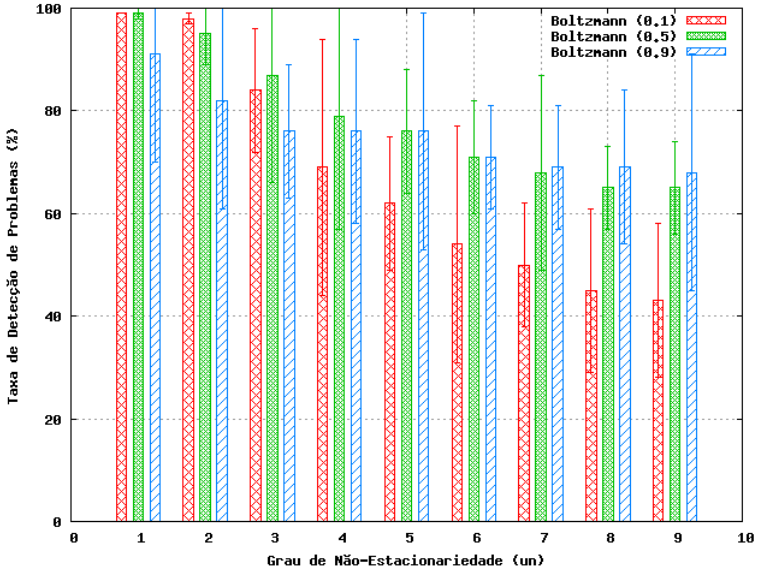
Fazendo uma analogia ao cenário de redes real e usando somente as políticas clássicas do aprendizado por reforço, o fator 0.9 é o único que permite ter resultados mínimos aceitáveis, estipulados por esse trabalho, que correspondem ao índice de detecção igual a 65%. Observe que a partir do grau de não estacionariedade igual a 3, tem-se uma perda considerável de detecções, o que pode ser maléfica no caso em que essa venha a ocorrer exclusivamente para uma determinada situação. Observa-se, ainda, uma grande variabilidade nos resultados das detecções, o que se justifica pelo caráter aleatório empregado no processo de escolha das políticas ϵ-Gulosa e *Boltzmann*.

Em contrapartida, a solução baseada na proposta de Bloqueio Temporário alcança índices de detecção acima de 90%, o que é considerado satisfatório nesse trabalho. Como pode ser visualizado na figura 6.3(c), esse método possui uma certa consistência no resultado de detecção independente da variabilidade de comportamento presente. Essa estabilidade no percentual de detecção de comportamento explica-se pelo não reuso imediato de estados visitados e penalizados recentemente pelo processo de escolha. Assim, estados que estão suspensos temporariamente não são opções válidas de escolha, o que automaticamente acaba dando chances a outros estados, priorizando aqueles que apresentam maior *valor estado-ação* na seqüência e, assim, sucessivamente.

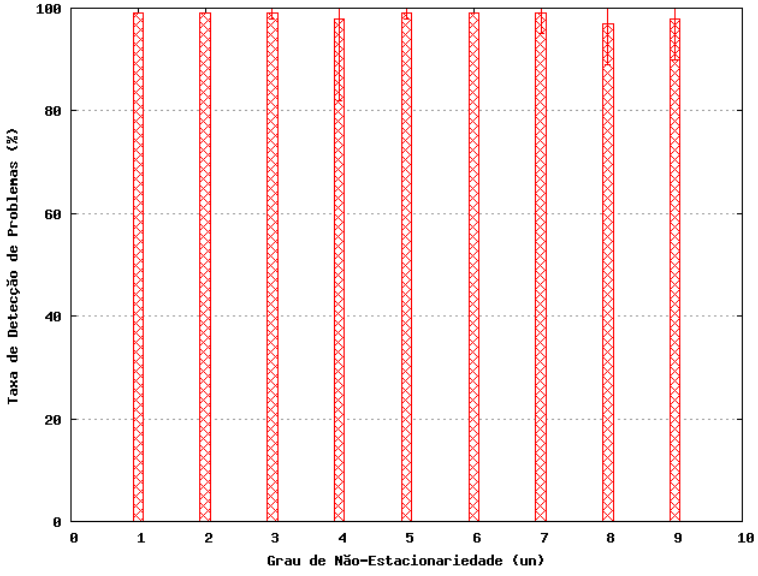
Em síntese, comparando as técnicas tradicionais ϵ-*Gulosa* e *Boltzmann* com a técnica



(a)



(b)



(c)

Figura 6.3: Teste de detecção de situações (α= 0,1)

proposta de Bloqueio Temporário, essa última pode ser considerada a melhor estratégia a ser utilizada em ambientes não estacionários. No entanto, esse modelo não foi testado em cenários contendo situações intermitentes de longa e de curta duração. No caso da ocorrência de situações acontecerem em períodos de tempos mais espaçados e com longa durabilidade, o aprendizado do sistema será mais lento, porém garante-se a detecção. Já na presença de comportamentos com curta duração ou rajada, o sistema apresentará uma aprendizagem parcial, uma vez que o tempo de ocorrência da situação é menor do que o tempo necessário para detectá-la. Dessa maneira, o sistema pode não alcançar um estado folha, o que implica em não garantir a exploração por completo do conjunto de estados possíveis.

6.3 Eficiência no Processo de Detecção de Situações

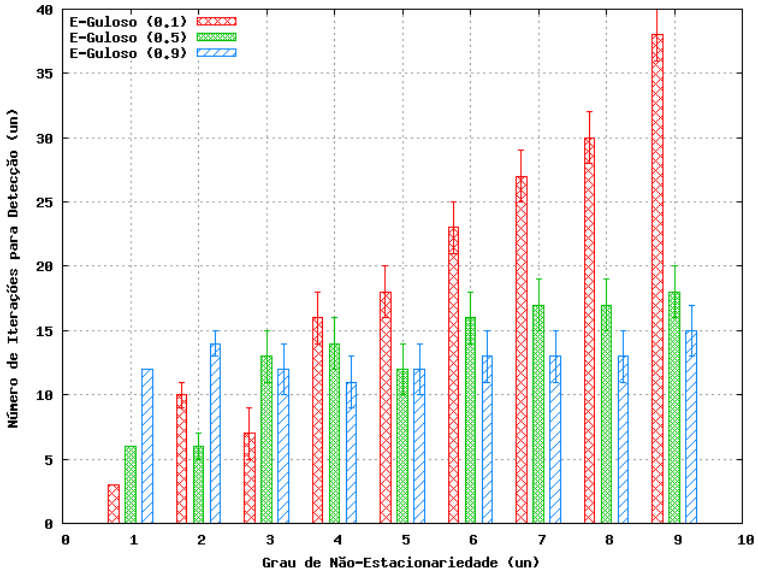
Na sub-seção anterior foram realizados experimentos que tiveram como objetivo a avaliação da eficácia das políticas empregadas. Esses experimentos resultaram na construção de diferentes árvores cada qual tendo, a seus estados, *valores estado-ação* próprios, dependendo da política e do fator de influência usados. Estas várias árvores treinadas foram usadas como entradas para a realização de experimentos relacionados à investigação da eficiência das políticas na detecção de situações.

Como medida de eficiência, no contexto desse trabalho, quantificou-se o número de iterações realizadas pelo algoritmo *Q-Learning* segundo as três políticas ϵ-Gulosa, *Boltzmann* e Bloqueio Temporário para detectar os comportamentos anômalos. Para cada grau de não estacionariedade *i* ($0< i <10$) foram conduzidos experimentos isolados para medir a quantidade de iterações necessárias para detectar as *i* situações, repetindo-se 30 vezes. Com essa repetição, pretendeu-se garantir uma boa representatividade amostral. Sobre essa quantia total, fez-se a média para cada grau de não estacionariedade. Esses experimentos ainda foram repetidos para diferentes fatores de influência (0.1, 0.5 e 0.9), mantendo o mesmo tempo de suspensão anteriormente definido e a taxa de aprendizado α igual a 0.1.

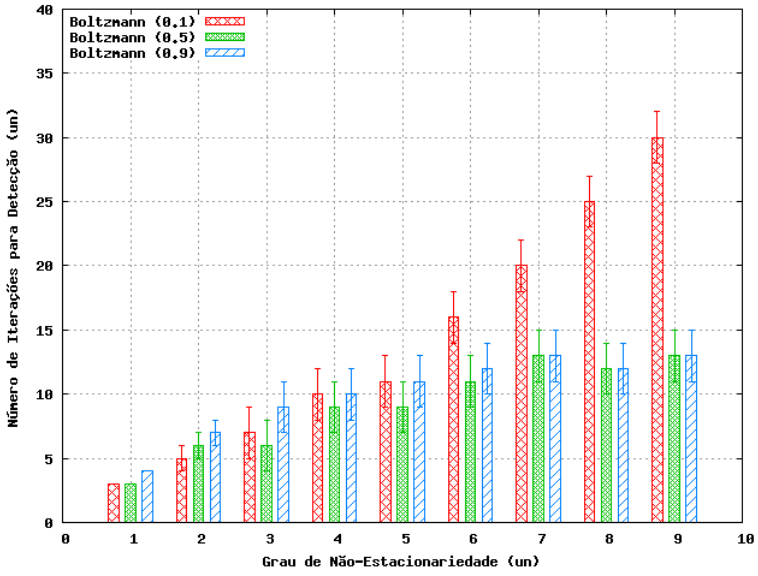
Conforme o gráfico ilustrado na figura 6.4(a), pode-se constatar que o método ϵ-Guloso, contendo o fator de influência igual a 0.9 apresenta os melhores resultados para grau de não estacionariedade acima de 2. Para esse método, quanto maior for o caráter aleatório na escolha do próximo estado, maior será a chance de detectar a situação por meio de uma quantidade menor de iterações. O segundo melhor resultado é alcançado pelo fator 0.5 e, o pior, pelo 0.1. Tal pode ser explicado pela visitação de todos os estados da árvore pelo fator 0.9 e, alguns deles, poderiam não ter sido alcançados pelos fatores 0.1 e 0.5.

O gráfico da figura 6.4(b) ilustra os resultados dos experimentos obtidos usando a política *Boltzmann*. Nesse gráfico, observa-se certo equilíbrio na quantidade média de iterações necessárias para detecção de situações entre os fatores de influência iguais a 0.5 e 0.9. Até o grau de não estacionariedade igual 6, prevalecem os melhores resultados para o fator 0.5. Após esse grau, ambos fatores apresentam quantia média de iterações equivalentes. O fator de influência igual a 0.1 repetiu o mesmo desempenho ruim do que o método ϵ-Guloso, embora, comparativamente, menos iterações foram necessárias para se detectar um comportamento.

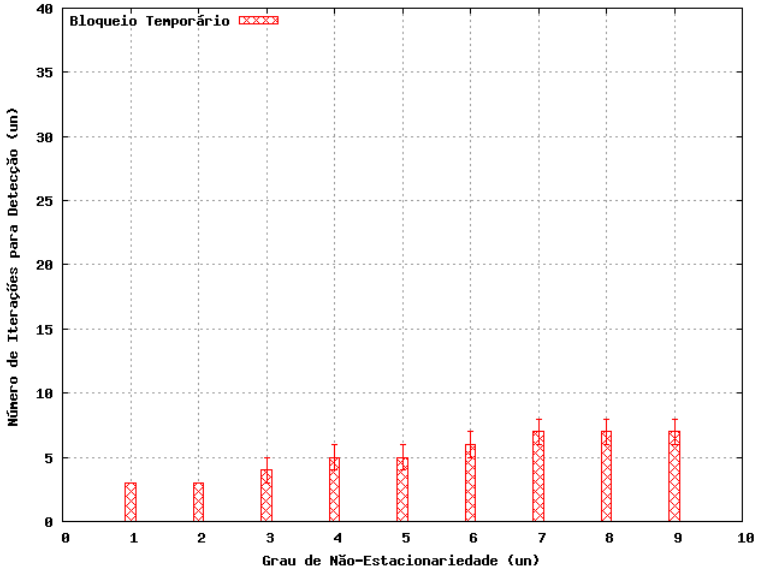
Por fim, o melhor resultado obtido pode ser visualizado na figura 6.4(c), que corresponde à política de bloqueio temporário. Repare que a quantidade média de iterações necessárias é crescente, embora permaneça abaixo de 10 unidades. Essa quantia é signifi-



(a)



(b)



(c)

Figura 6.4: Média da quantidade de iterações necessárias para detecção de situações(α= 0,1)

cativamente inferior as médias de iterações exigidas pelas políticas *Boltzmann* e ϵ-Gulosa. Esse resultado reafirma o apresentado em 6.3(c) e justifica o alto índice de detecção de comportamentos, devido à suspensão temporária de estados penalizados. Dessa forma, não se escolhe o próximo estado somente pelo maior valor-estado ação, como é empregado na maior parte das vezes pelas políticas tradicionais. Logo, a suspensão temporária do estado penalizado aumenta o caracter exploratório no processo de seleção de estados a serem visitados.

Considere um cenário real de monitoração de redes, na qual tem-se como aceitável a realização de requisições aos objetos gerenciados a cada 30 segundos, conforme sugerido em (WALDBUSSER, 2000), e a identificação da presença de um comportamento anômalo em tempo inferior a 5 minutos. A viabilidade de uso dessas diferentes políticas nesse ambiente dinâmico implica em garantir que as detecções sejam realizadas com a quantia máxima de iterações iguais a 10 unidades. Retornando aos gráficos da figura 6.4, pode-se verificar que a política ϵ-Gulosa atende esse requisito para os fatores de influência iguais a 0.1 e 0.5, considerando o grau de não estacionariedade variando até 3 e 2, respectivamente. Já o método *Boltzmann* consegue cumprir esse objetivo para todos os fatores de influência testados até o grau de não estacionariedade 4. Para esse cenário, a única forma de alcançar a identificação de todas as nove situações em tempo hábil baseia-se no emprego do método de bloqueio temporário proposto.

7 CONCLUSÃO

A automatização de procedimentos na gerência de redes está se tornando cada vez mais necessária, uma vez que cresce exponencialmente a complexidade advinda da incorporação de novas tecnologias e serviços na rede. Em breve, os administradores não serão capazes de lidar com todas as dificuldades introduzidas ao gerenciamento de redes. Em virtude dessa limitação, é crucial buscar alternativas para desonerar os administradores de suas atividades de monitoramento e configuração para que possam concentrar esforços no planejamento do uso da rede. Ao longo dos anos, diversas tentativas de agregar técnicas de Inteligência Artificial às soluções de gerenciamento têm sido propostas com o objetivo de reduzir o esforço humano.

A área da Computação Autonômica, particularmente, tem-se beneficiado das técnicas de IA, uma vez que as incorpora nas entidades autonômicas a fim de assegurar as propriedades de configuração, otimização, reparo e proteção. Os sistemas baseados nessa abordagem são formados por elementos autonômicos, os quais são responsáveis por gerenciar seu próprio comportamento e suas interações com o ambiente. Assim, a autogerenciabilidade do sistema emerge dessas ações recíprocas entre os componentes. A grande vantagem dessa metodologia consiste na decomposição do problema em vários elementos autonômicos, os quais utilizam técnicas conjuntas da IA para representar o conhecimento do domínio, realizar o planejamento de ações, enfim, reagindo perante modificações ocorridas no ambiente.

Logo, as tecnologias de auto-gerenciamento se mostram capazes de lidar com a evolução contínua da natureza dos atuais sistemas de gerência e dos processos de monitoração de informações. Na busca por mecanismos de IA, os quais pudessem prover a adaptabilidade necessária ao ambiente para compor um elemento autonômico, a abordagem de aprendizado por reforço destacou-se por apresentar essa característica através do aprendizado por tentativa e erro. Dessa forma, o ambiente pode evoluir para estados que não apresentam ações correspondentes e o sistema deverá explorar o conjunto de ações de forma a encontrar a mais adequada entre elas. Como o cenário de monitoração de redes é dinâmico, envolvendo muitas alterações de comportamento, o método de aprendizado por reforço consegue suprir essa necessidade de constante adaptação.

A partir dessas considerações, as principais contribuições desse trabalho, como anteriormente apresentadas, são listadas a seguir.

- Proposta de um modelo para representação, em estados, dos objetos gerenciáveis relacionados à identificação de comportamentos anômalos na rede.
- Exploração das políticas tradicionais ϵ-Gulosa e *Boltzmann*.
- Proposta e uso de uma nova política intitulada Bloqueio Temporário.

- Desenvolvimento de protótipo de um elemento autonômico, baseado em aprendizado por reforço, a fim de conferir adaptação ao processo de monitoração.

Durante a fase de elaboração da solução exposta no capítulo 5 existiram algumas dificuldades, as quais foram ultrapassadas para alcançar os resultados descritos no capítulo 6. Inicialmente, foi necessário definir uma estrutura de estados de monitorações em forma de árvore a fim de preservar as relações de interdependência entre os objetos de monitoração. Entretanto, os métodos de aprendizado por reforço tradicionalmente usufruem de tabelas para representação de estados, o que não satisfaz as necessidades do cenário de monitoração. Outra dificuldade encontrada consistiu na captura de dados de um ambiente real para poder avaliar a solução proposta. Geralmente, não há garantias que um ambiente real irá apresentar todos os comportamentos contendo as unidades de monitoração necessárias para serem detectados. Isso foi contornado com a criação de um sistema de apoio para a geração e emulação de todos os comportamentos necessários para o processo de avaliação.

Os resultados obtidos pela implementação e avaliação desse modelo, tendo em vista a árvore de estados exemplo utilizada, demonstraram que as abordagens tradicionais ϵ-Gulosa e *Boltzmann* garantem índices de detecção de comportamentos entre 65% e 70% para ambientes cuja não estacionariedade é elevada. Em contrapartida, a política de bloqueio temporário alcançou resultados superiores, com taxa de detecção acima de 90% para todos os graus de não estacionariedade. Além disso, o bloqueio temporário apresentou certa estabilidade do percentual de detecção de comportamento independente da variabilidade do grau de não estacionariedade e exigiu uma baixa quantia de iterações para identificação das situações. Essas conclusões foram condensadas na forma de um artigo, intitulado *How Much Management is Management Enough? Providing Monitoring Processes with Online Adaptation and Learning Capability* (COELHO; GASPARY; TAROUCO, 2009), submetido como *short paper* em uma das principais conferências de Gerência de Redes de Computadores (padrão Qualis A internacional): *IFIP/IEEE International Symposium on Integrated Network Management*.

Como trabalhos futuros, têm-se a necessidade de ampliar o cenário de teste para representar situações intermitentes de longa e de curta duração a fim de obter uma melhor avaliação do modelo. Pretende-se, também, avaliar o modelo perante dados de monitoração reais com o objetivo de verificar o tempo de aprendizado de acordo com os diferentes tipos de redes como, por exemplo, industriais, acadêmicas e domésticas. Outro trabalho a ser realizado, relacionado ao modelo, consiste na agregação de mais estados e na modificação de suas interligações para verificar o impacto da disposição dos estados em diferentes árvores na eficiência da solução. A respeito da regra de avaliação presente em cada estado, poder-se-ia encontrar alguma função polinomial capaz de expressar a relação existente entre os diferentes objetos de monitoração que o compõe, com o objetivo de diminuir o conhecimento inicial no modelo.

REFERÊNCIAS

ACCA: Autonomic Communication: Coordination Action Homepage. Disponível em: <http://www.autonomic-communication.org/projects/acca/>. Acesso em: 20 dez. 2007.

AGOULMINE, N. et al. Challenges for Autonomic Network Management. In: IEEE INTERNATIONAL WORKSHOP ON MODELLING AUTONOMIC COMMUNICATIONS ENVIRONMENTS, 1., 2006, Dublin, Ireland. **Proceedings...** [S.l.]: IEEE Computer Society, 2006. p.47–67.

ARORA, A. et al. Web Services for Management (WS-Management). Desktop Management Task Force (DMTF), DSP0226_1.0.0, Febrary 2008. Disponível em: <http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf>. Acesso em: 4 out. 2008.

BULLARD, V.; VAMBENEPE, W. **Web Services Distributed Management**: management using web services (muws 1.1) part 1. [S.l.]: Organization for the Advancement of Structured Information Standards (OASIS), 2006. Disponível em: <http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf>. Acesso em: 4 out. 2008.

BULLARD, V.; VAMBENEPE, W. **Web Services Distributed Management**: management using web services (muws 1.1) part 2. [S.l.]: Organization for the Advancement of Structured Information Standards (OASIS), 2006. Disponível em: <http://docs.oasis-open.org/wsdm/wsdm-muws2-1.1-spec-os-01.pdf>. Acesso em: 4 out. 2008.

COELHO, J. O.; GASPARY, L. P.; TAROUCO, L. M. R. How Much Management is Management Enough? Providing Monitoring Processes with Online Adaptation and Learning Capability. Aceito para o IFIP/IEEE International Symposium On Integrated Network Management, 2009, New York.

DARPA: Defense Advanced Research Projects Agency Homepage. Disponível em: <http://www.darpa.mil/ipto/programs/acip/acip.asp>. Acesso em: 20 dez. 2007.

DRESSLER, F. Bio-inspired promoters and inhibitors for self-organized network security facilities. In: INTERNATIONAL CONFERENCE ON BIO INSPIRED MODELS OF NETWORK, INFORMATION AND COMPUTING SYSTEMS, BIONETICS, 1., 2006, New York, NY, USA. **Proceedings...** New York: ACM, 2006. p.25.

DTMF: Homepage. Disponível em: <http://www.dmtf.org/>. Acesso em: 10 out. 2008.

IETF: Homepage. Disponível em: <www.ietf.org/>. Acesso em: 10 out. 2008.

ITU: Homepage. Disponível em: <http://www.itu.int/>. Acesso em: 10 out. 2008.

JAVA: Homepage. Disponível em: <http://java.sun.com/>. Acesso em: 20 nov. 2007.

KEPHART, J. O. Research challenges of autonomic computing. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE, 27., 2005, Taipei, Republica da China. **Proceedings...** Skokie: Knowledge Systems Institute, 2005. p.15–22.

KEPHART, J. O.; CHESS, D. M. The Vision of Autonomic Computing. **Computer**, Los Alamitos, CA, USA, v.36, n.1, p.41–50, 2003.

MACHIDA, F.; KAWATO, M.; MAENO, Y. Adaptive Monitoring for Virtual Machine Based Reconfigurable Enterprise Systems. In: INTERNATIONAL CONFERENCE ON AUTONOMIC AND AUTONOMOUS SYSTEMS, ICAS, 3., 2007, Atenas, Grécia. **Proceedings...** [S.l.]: IEEE Computer Society, 2007. p.8.

MCCLOGHRIE, K.; ROSE, M. T. **Management Information Base for Network Management of TCP/IP-based internets**:MIB-II. United States of America: RFC Editor, 1991.

NETBEANS: Homepage. Disponível em: <http://www.netbeans.org/>. Acesso em: 20 nov. 2007.

NUNES, C. M. **Um discriminador inteligente de eventos de rede para o ambiente CINEMA**. 1997. Dissertação (Mestrado em Ciência da Computação)-Instituto de Informática, UFRGS, Porto Alegre.

OMG: Homepage. Disponível em: <http://www.omg.org/>. Acesso em: 10 out. 2008.

PRAS, A. et al. Comparing the Performance of SNMP and Web Services-Based Management. **IEEE Transactions on Network and Service Management**, [S.l.], p.72–82, 2004.

PRIETO, A. G.; STADLER, R. A-GAP: an adaptive protocol for continuous network monitoring with accuracy objectives. **IEEE Transactions on Network and Service Management**, [S.l.], v.4, n.1, p.2–12, 2007.

RUSSEL, S.; NORVIG, P. **Artificial Intelligence**: modern approach. 2nd ed. New York, USA: Prentice Hall, 2005.

SALEHIE, M.; TAHVILDARI, L. Autonomic computing: emerging trends and open problems. **SIGSOFT Softw. Eng. Notes**, New York, NY, USA, v.30, n.4, p.1–7, 2005.

SCHÖNWÄLDER, J. et al. SNMP Traffic Analysis: approaches, tools, and first results. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, 10., 2007, Munique, Alemanha. **Proceedings...** [S.l.]: IEEE Computer Society, 2007. p.323–332.

SNMP4J: Homepage. Disponível em: <http://www.snmp4j.org/>. Acesso em: 20 nov. 2007.

STALLINGS, W. **SNMP, SNMPv2, and RMON**: practical network management. 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1997.

STRASSNER, J.; AGOULMINE, N.; LEHTIHET, E. FOCALE - A Novel Autonomic Networking Architecture. In: LATIN AMERICAN AUTONOMIC COMPUTING SYMPOSIUM, LAACS, 1., 2006, Campo Grande, MS, Brasil. **Proceedings...** [Campo Grande]: SBC, 2006. p.48–60.

SUPERCSV: Homepage. Disponível em: <http://supercsv.sourceforge.net/>. Acesso em: 20 nov. 2007.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning**: an introduction. [S.l.]: MIT Press, 1998.

TESAURO, G. Online Resource Allocation Using Decompositional Reinforcement Learning. In: AAAI, 2005, Pittsburgh, USA. **Proceedings...** [S.l.]: AAAI Press / The MIT Press, 2005. p.886–891.

TESAURO, G. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In: INTERNATIONAL CONFERENCE ON AUTONOMIC COMPUTING, ICAC, 3., 2006, Dublin, Ireland. **Proceedings...** [S.l.]: IEEE Computer Society, 2006. p.65–73.

TESAURO, G. Reinforcement Learning in Autonomic Computing: a manifesto and case studies. **IEEE Internet Computing**, [S.l.], v.11, n.1, p.22–30, 2007.

TMF: Homepage. Disponível em: <http://www.tmforum.org/>. Acesso em: 10 out. 2008.

WHITESON, S.; STONE, P. Adaptive Job Routing and Scheduling. **Engineering Applications of Artificial Intelligence**, [S.l.], v.17, n.7, p.855–69, October 2004. Special issue on Autonomic Computing and Automation.

XSTREAM: Homepage. Disponível em: <http://xstream.codehaus.org/>. Acesso em: 20 nov. 2007.

WALDBUSSER, S. **Network Monitoring Management Information Base**. United States of America: RFC Editor, 2000.