2
Fundamentos e Trabalhos Relacionados

Esta tese aborda o gerenciamento autonômico de sistemas distribuídos. O foco é a caracterização de problemas de desempenho de aplicações distribuídas construídas sobre tecnologias de *middleware*. Para efeito de experimentação e validação de propostas apresentadas neste trabalho, adotamos um *middleware* baseado em componentes de software. Atualmente, tecnologias de componentes são uma das opções mais utilizadas para o desenvolvimento de aplicações complexas que exigem grande dinamismo e flexibilidade. Este capítulo provê fundamentos relevantes para o tema de pesquisa em questão. São discutidos conceitos relacionados a gerenciamento de desempenho, tecnologia de componentes e computação autonômica. Baseado nesses conceitos, o capítulo discute, subsequentemente, importantes modelos de caracterização de desempenho encontrados na literatura. Finalmente, o capítulo conclui com algumas questões remanescentes, para as quais os trabalhos atuais existentes ainda não apresentam soluções consolidadas.

2.1
Gerenciamento de Desempenho

Para muitas aplicações, requisitos de qualidade de serviço são tão importantes quanto requisitos funcionais. Nessas aplicações, a não satisfação dos requisitos de qualidade pode comprometer o comportamento esperado do sistema, levando-o ao não cumprimento de seus objetivos. Ghezzi et al. (Ghezzi et al., 1991) classificam os requisitos de qualidade de software em duas categorias principais: os externos ou observados em tempo de execução e os internos ou observados em tempo de projeto. Requisitos de qualidade observados em tempo de execução incluem, por exemplo, desempenho e confiabilidade, enquanto atributos observados em tempo de projeto abrangem requisitos como manutenibilidade, reusabilidade, portabilidade, dentre outros. Esta tese está relacionada com os requisitos da primeira categoria. O foco é o estudo de abordagens para o gerenciamento do desempenho de aplicações distribuídas baseadas em *middleware*.

O glossário de terminologias de Engenharia de Software do IEEE (IEEE,

1992) define, genericamente, o termo desempenho como o nível de satisfação atingido por um sistema computacional ao executar suas funcionalidades dentro de certas restrições (como velocidade, acurácia ou uso de memória). De acordo com essa definição, parâmetros ou métricas devem ser definidos para especificar, mensurar e avaliar o desempenho de um sistema. Algumas métricas comumente usadas para esse propósito incluem: *i*) tempo de resposta, o tempo decorrido desde o momento em que uma requisição tem início até o momento em que ela é completamente atendida; *ii*) latência, o tempo que leva um sistema para reagir a uma requisição de serviço; e *iii*) vazão, o número de requisições ou eventos processados por um sistema em um período de tempo.

O desempenho dos sistemas computacionais pode ser influenciado por fatores como carga, o tipo e a quantidade de recursos disponíveis e a forma como esses recursos são alocados. Em geral, a carga de um ambiente ou sistema pode ser caracterizada em termos da frequência e do tipo de requisições ou através de níveis de utilização de recursos. Recursos de software incluem *threads*, processos e conexões. Recursos de hardware abrangem CPU, memória, dispositivos de entrada e saída e recursos de rede.

Considerando que o desempenho dos sistemas é afetado pelos fatores descritos acima, métodos de gerenciamento são usados para tratar e otimizar métricas de desempenho. Em geral, tais métodos abrangem atividades de previsão ou estimação de valores para métricas de desempenho, bem como o diagnóstico da natureza de um problema, quando este é detectado. Métodos de gerenciamento de desempenho são aplicados em diferentes estágios do ciclo de vida de um sistema. Em tempo de projeto, métodos de previsão e avaliação de desempenho podem ser aplicados à especificação, arquitetura, projeto e implementação do sistema. Há uma grande quantidade de trabalhos com foco nesses tipos de métodos que exploram desde notações formais, como grafos e linguagens arquiteturais (Leclercq et al., 2007), até extensões em notações como UML ou RM-ODP (OMG, 2005; Dimitrov et al., 2002; Cortellessa et al., 2007). Métodos de gerenciamento de desempenho também podem ser aplicados para sistemas em execução, uma vez que esse é um requisito de qualidade externo e, portanto, observado apenas em tempo de execução. Sob esse aspecto, há diversos trabalhos que implementam mecanismos de monitoramento e coleta de dados em diferentes tipos de sistemas e em diferentes camadas: aplicações EJB (Mos e Murphy, 2002), servidores de aplicação (HP, 2010), *middleware* (Debusmann et al., 2002; Marchetti, 2001), máquinas virtuais (Sun, 2010), sistemas operacionais (Godard, 2010) e redes (Galstad, 2010)). Para alguns tipos de sistemas em que o comportamento completo requerido é conhecido apenas em tempo de execução, como é o caso de sistemas baseados em

middleware, esse é o momento mais oportuno para o controle do desempenho. Por esse motivo, este trabalho utiliza métodos de gerenciamento aplicados em tempo de execução. A seguir, são apresentados alguns fundamentos sobre a tecnologia de *middleware* usada neste trabalho.

2.2 Sistemas Baseados em Componentes de Software

Programação baseada em componentes de software é um paradigma emergente. Considerado por muitos a evolução da programação orientada a objetos, esse novo paradigma tem sido extensivamente aplicado na construção de sistemas de grande escala, os quais são construídos a partir da integração de componentes de software já existentes. O termo *componente de software* é usado em diversos contextos e com diferentes significados. Neste trabalho, adotamos a definição fornecida por Clemens Szyperski (Szyperski, 2002). De acordo com essa visão, um componente de software é uma unidade de composição com interfaces especificadas contratualmente e dependências contextuais (como interfaces requeridas e plataformas suportadas) explícitas. Além de unidades de composição, componentes de software são unidades de implantação. Isto significa que componentes podem ser implantados independentemente, sendo seu uso possível inclusive para composição por terceiros.

De forma geral, a definição acima estabelece duas características principais para componentes de software: *i*) componentes devem ser autocontidos, ou seja, devem ser separados de outros componentes e do seu ambiente de execução; e *ii*) componentes devem especificar claramente o que são capazes de oferecer e quais são suas dependências de contexto. Devido a essas características, a abordagem de desenvolvimento por componentes de software aumenta potencialmente a reutilização de módulos de software e a flexibilidade de composição de diferentes módulos. Assim, tecnologias baseadas nessa abordagem de desenvolvimento tornam-se soluções atraentes para aplicações complexas e que demandam flexibilidade.

No entanto, para que componentes desenvolvidos independentemente possam cooperar de forma útil, eles devem seguir um padrão ou protocolo comum. As plataformas de *middleware* baseadas em componentes de software estabelecem esse padrão, determinando as condições para a instanciação de componentes e o controle da interação entre eles. Quando uma aplicação é construída sobre uma tecnologia particular de *middleware* baseada em componente, todos os componentes de software usados na montagem da aplicação, também conhecidos como componentes de aplicação, seguem o mesmo protocolo definido pelo *middleware*, podendo, dessa forma, interoperar. Java 2

Platform Enterprise Editions (J2EE) (Oracle, 2011), Corba Component Model (CCM) (Wang et al., 2001), Fractal (OW2Consortium, 2011) e .NET (Microsoft, 2011) são alguns exemplos de plataformas de *middleware* baseadas em componentes.

É comum agrupar em três partes as funcionalidade providas por um *middleware* baseado em componente. A primeira parte, denominada de núcleo, provê funcionalidades básicas de invocação e comunicação. Isto inclui a localização do componente provedor, o transporte de requisições e respostas e o empacotamento/desempacotamento de parâmetros. A segunda parte, denominada serviços comuns, provê os serviços de infraestrutura requeridos pelos diversos componentes de aplicação. Finalmente, a terceira parte, denominada contêiner ou cápsula, provê um contexto de execução para os componentes de aplicação. É responsabilidade do contêiner a instanciação dos componentes de aplicação, o controle do ciclo de vida desses componentes e o acesso aos serviços comuns. Em geral, esses serviços tornam os componentes de aplicação mais independentes do ambiente de execução. Detalhes sobre a tecnologia de componentes usada neste trabalho são apresentados no Capítulo 3.

Em resumo, Bouziane et al. citam os seguintes elementos como definições essenciais em um *middleware* baseado em componentes (Bouziane et al., 2006):

1. Composição: um componente pode ser agregado a outros componentes, inclusive de terceiros. Esta composição é possível graças a interfaces bem definidas que permitem a interação entre componentes. Alguns contratos são vinculados a essas interfaces e devem ser aceitos para o estabelecimento da interação.

2. Portas: para interagir com outros componentes, um componente define interfaces externas denominadas portas. Uma porta é um artefato de programação ao qual uma interface pode ser vinculada. Portas são categorizadas em dois tipos: cliente (porta para serviços requeridos) ou servidor (porta provedora de serviço). A interação entre dois componentes é então concretizada conectando-se a porta cliente de um componente com a porta servidor de outro com tipo compatível.

3. Separação de interesse: modelos de componentes têm por objetivo separar interesses funcionais dos não funcionais. O objetivo é não sobrecarregar o desenvolvedor de aplicações com requisitos não funcionais. Estes são ativados durante a configuração da aplicação e são fornecidos pela infraestrutura na qual os componentes são executados.

4. Montagem: a fase de montagem gera uma especificação das instâncias de componentes de aplicação e suas interconexões. A montagem pode ser descrita usando uma linguagem de descrição de arquitetura (ADL) ou usando composição em tempo de execução.

5. Implantação: um componente é uma unidade de implantação. Portanto, ele contém uma implementação e restrições associadas como, por exemplo, requisitos de sistema operacional, processador e memória. Essas propriedades ajudam uma ferramenta de implantação a tomar decisões sobre os recursos a serem alocados para a execução de um componente.

Gerenciamento de Desempenho em Middleware Baseado em Componentes

Existem diversos desafios associados à gerência de desempenho de sistemas baseados em componentes. As características particulares desses sistemas, discutidas na Seção anterior, diminuem a eficiência dos métodos de gerenciamento aplicados em tempo de projeto. Dentre essas características, a separação de interesses pode ser um fator de dificuldade para a compreensão do comportamento desses sistemas. Em tempo de projeto, componentes de aplicação são construídos com base em componentes de infraestrutura (*middleware*) e conectados a estes apenas em tempo de execução. O desempenho dos componentes de aplicação depende, portanto, do desempenho dos componentes de infraestrutura, cujo comportamento varia de acordo com fatores como tecnologia e implementação usadas. Esse acoplamento entre os elementos da aplicação e os de infraestrutura aumenta significativamente a complexidade de compreender os efeitos de diferentes decisões arquiteturais. Enquanto essa situação pode acontecer em outros tipos de sistemas baseados em *middleware*, em que componentes são representados por aplicações inteiras, a frequência e a extensão de tais acoplamentos é tipicamente maior em sistemas baseados em componentes.

Outra característica dos sistemas baseados em componentes que pode dificultar a compreensão do comportamento desses sistemas em tempo de projeto é a dependência de contexto explícita. Essa característica facilita a troca de módulos de software em tempo de execução, tornando as tecnologias baseadas em componentes soluções inerentemente dinâmicas. Como consequência, a fim de atender contexto específicos, a configuração de uma aplicação em tempo de execução pode variar significativamente daquela idealizada em tempo de projeto. Por outro lado, é uma tarefa difícil, senão inviável, tentar prever, em tempo de projeto, todas as configurações possíveis para os componentes de aplicação.

Em resumo, essas dificuldades tornam inviável o uso de métodos estáticos de gerenciamento para o controle de sistemas baseados em componentes. Ao contrário, métodos de gerenciamento que se baseiam em processos contínuos e dinâmicos são opções mais apropriadas nessas condições. No entanto, uma consequência direta dessa forma de gerenciamento é que tarefas complexas de otimização e configuração passam a ser executadas repetidamente em tempo de execução. Essas, porém, são tarefas difíceis de executar manualmente. Portanto, há uma grande necessidade de tornar esse processo de gerenciamento automatizado e, se possível, diminuir a intervenção humana sobre ele.

2.3 Computação Autonômica

Computação Autonômica[1] é um termo introduzido pela IBM e denota o conjunto de esforços para tratar o problema da complexidade de gerenciamento dos sistemas computacionais, permitindo que os mesmos se encarreguem do próprio gerenciamento. Esse controle, no entanto, deve ser feito segundo políticas administrativas de alto nível (Kephart e Chess, 2003)(Ganek e Corbi, 2003). Sob esse aspecto, sistemas autonômicos devem ser capazes de manter e ajustar sua operação em resposta a mudanças internas em requisitos funcionais ou em função de modificações externas motivadas por requisitos de qualidade, como reparo, otimização ou proteção. De fato, Ganek et al. (Ganek e Corbi, 2003) definem um sistema autonômico ideal como aquele que trabalha ininterruptamente, promovendo reconfigurações e otimizações quando necessário.

Para atingir o objetivo descrito acima, algumas arquiteturas de software para computação autonômica foram propostas. Em geral, essas arquiteturas apresentam soluções que executam continuamente um laço de controle (Figura 2.1) envolvendo as atividades descritas a seguir:

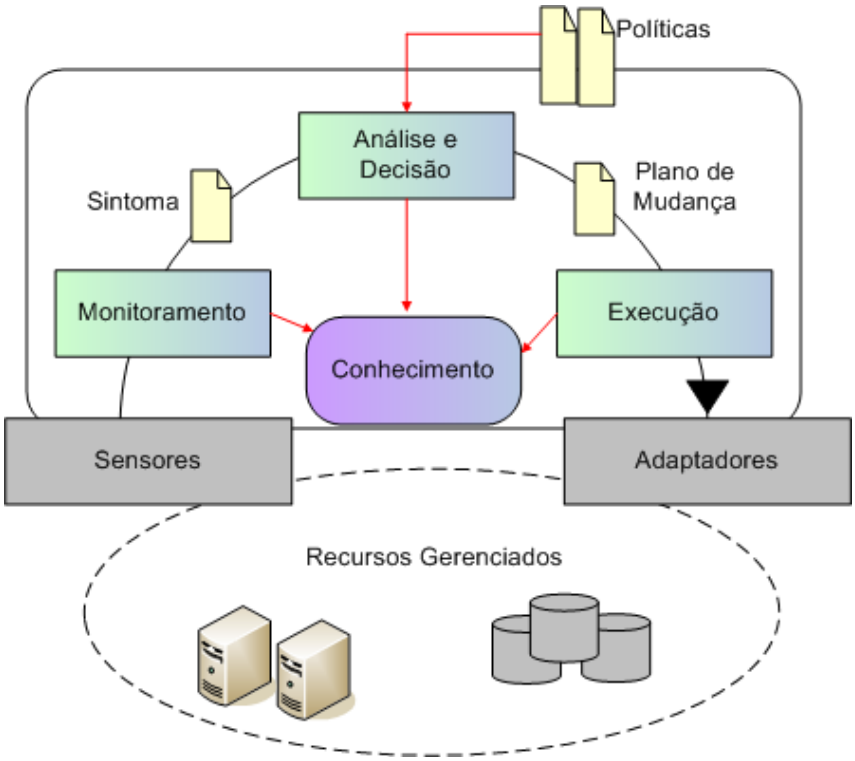
- monitoramento ou medição: função que coleta, agrega, correlaciona e filtra dados sobre recursos gerenciados. Recursos gerenciados incluem servidores, unidades de armazenamento, banco de dados, servidores de aplicação, serviços, aplicações, etc. Dados coletados incluem: informações de topologia, eventos, métricas, propriedades de configuração, etc.
- análise e decisão: a função de análise examina os dados coletados e determina se devem ser feitas mudanças sobre as políticas ou estratégias correntes. Essa tomada de decisão assegura convergência em relação

[1]Do termo inglês *Autonomic Computing*. Os termos *Computação Autônoma* e *Computação Autonômica* são ambos usadas na literatura. Neste trabalho, adotamos o termo *Computação Autonômica*.

a valores limiares de parâmetros como desempenho, disponibilidade e segurança.

– controle e execução: a função de controle escalona e executa as mudanças identificadas como necessárias pela função de análise e decisão. Mudanças são executadas na forma de ações que modificam o estado de um ou mais recursos gerenciados.

Figura 2.1: Ciclo de gerenciamento em arquiteturas autonômicas. Baseada em (Parashar e Hariri, 2007)



Como o laço de controle descrito acima é executado repetidamente, os efeitos de uma ação tomada pela função de controle e execução no final de um ciclo podem ser medidos e analisados nos ciclos subsequentes.

Resumindo, o termo computação autonômica é empregado para representar um conjunto de tecnologias integradas que tornam possíveis o autogerenciamento dos sistemas. Uma solução de autogerenciamento inclui aspectos de autocura (*self-healing*), auto-otimização (*self-optimizing*), autoproteção (*self-protecting*) e autoconfiguração (*self-configuring*). Pesquisas em cada uma dessas áreas contribuem para a realização da visão proposta pela computação autonômica. Nesse contexto, este trabalho está alinhado com essa iniciativa, uma vez que ele concentra-se no aspecto de auto-otimização de sistemas baseados em componentes. É importante ressaltar, no entanto, que o escopo deste

trabalho envolve apenas as atividades de monitoramento e análise de dados, com foco principal no uso desses dados na construção de modelos para caracterizar o comportamento do desempenho das aplicações. As demais atividades do laço de controle não são abordadas neste trabalho. A seguir é apresentada uma revisão sobre modelagem de desempenho em sistemas autonômicos. Uma discussão mais profunda sobre computação autonômica, tratando conceitos básicos e infraestruturas, está disponível em (Correa e Cerqueira, 2009).

2.4 Trabalhos Relacionados

O problema da construção de sistemas distribuídos que atendam às expectativas de desempenho dos usuários tem sido abordado por diversos trabalhos. Esse é um problema cada vez mais comum, dada a complexidade e a escala dos sistemas atuais. Nesta seção, são apresentados alguns trabalhos que propõem soluções autonômicas para esse problema. Os trabalhos são agrupados em três categorias. A primeira categoria compreende os trabalhos que se concentram na construção de modelos de previsão ou estimação de problemas de desempenho. A segunda categoria é um subconjunto da primeira e abrange os trabalhos que focam a construção de modelos de previsão de problemas de desempenho pelo uso de abordagens de aprendizado *online*. Finalmente, a terceira categoria compreende os trabalhos que se dedicam ao diagnóstico das causas de um problema quando este é detectado.

2.4.1 Modelos para Previsão de Problemas de Desempenho

Em geral, a carga percebida por um sistema tende a variar dinamicamente e períodos curtos de flutuação são frequentes. Um grande desafio em gerenciamento de sistemas é fazer um bom uso dos recursos existentes a fim de tratar esses momentos de flutuação e manter a qualidade de serviço. Para tratar esse problema, modelos para previsão de comportamento são essenciais. Dois grupos de trabalhos se destacam nesse propósito.

O primeiro grupo abrange os trabalhos que usam modelos analíticos para previsões de problemas de desempenho. Dentre esses modelos, teoria de fila tem sido usada extensivamente para modelar o comportamento de aplicações Internet de camada única, como servidores HTTP, ou multicamadas (Zhu et al., 2001; Bennani e Menasce, 2005; Urgaonkar et al., 2008; Chandra et al., 2003; Abrahao et al., 2006; Almeida et al., 2006). Modelos de fila de camada única se concentram apenas na captura do desempenho do recurso de maior utilização, ou seja, a camada que representa o gargalo do sistema. Um exemplo

de trabalho que usa essa abordagem pode ser visto em (Villela et al., 2007), onde a camada de aplicação de um sistema para *e-comerce* foi modelada por uma fila M/GI/1/PS. Por outro lado, quando teoria de fila é aplicada para modelar aplicações multicamadas, cada camada é representada por uma fila. Essa foi a abordagem escolhida por Urgaonkar et al. em (Urgaonkar et al., 2005; Urgaonkar e Chandra, 2005). Esses trabalhos usam redes abertas e fechadas e um algoritmo de Análise do Valor Médio (*Mean Value Analysis* – MVA) para caracterizar o tempo de serviço médio dos servidores de cada camada e, assim, prever a capacidade de todo o sistema. Entretanto, uma limitação desses trabalhos é a suposição de que a carga é composta por um conjunto de requisições fixas. Uma abordagem mais realista e que leva em consideração cargas compostas por diferentes conjuntos de requisições é apresentada por Zhang et al. (Zhang et al., 2007; Zhang et al., 2008). Ao contrário do proposto em (Urgaonkar e Chandra, 2005), os autores não estimam o tempo médio de serviço de cada servidor, mas sim, caracterizam o custo individual de cada tipo de transação que pode compor uma carga. Para aproximar o custo de serviço de cada tipo de transação, os autores usam um método de regressão estatística. O custo de cada tipo de transação e a composição da carga são então usados para estimar o tempo de serviço de cada fila ou servidor, o qual passa a variar com a composição da carga. Usando uma rede fechada, a capacidade do sistema é calculada usando MVA.

Rede de Petri estocástica genérica (*Generalised Stochastic Petri Net* – GSPN) é outra abordagem analítica que aparece na literatura recente de gerenciamento de desempenho de sistemas distribuídos (Dingle et al., 2002; Haggarty et al., 2009). Nessa abordagem, redes de Petri tradicionais são estendidas para incorporar informações temporais e o modelo obtido é usado para análise de tempos de resposta. De fato, distribuições de tempo de resposta são geralmente as saídas do processo de análise. Essas distribuições são extraídas por métodos numéricos a partir de cadeias de Markov ou semi-Markov. Todavia, uma dificuldade dessa abordagem é o custo computacional, o qual se torna inviável para modelos com muitos estados.

Considerando ainda modelos analíticos, outra abordagem representativa que tem sido usada em trabalhos de gerenciamento de sistemas dinâmicos é teoria de controle, como pode ser visto em (Lu et al., 2001; Cervin et al., 2002; Lu et al., 2003; Blanquer et al.; Diao et al., 2002; Abdelzaher et al., 2002). Todos esses trabalhos usam um controle com realimentação para primeiramente observar o estado atual do sistema e, então, tomar alguma decisão para atingir a qualidade de serviço desejada. Uma extensão desse modelo clássico de controle foi proposta por Kandasamy et al. (Kandasamy et al., 2004) para oti-

mizar o gerenciamento de energia em sistemas computacionais. Nesse modelo, conhecido como *Limited Lookahead Control* (LLC), as ações de controle, que governam a operação do sistema, são obtidas otimizando o comportamento previsto (descrito por um modelo matemático) para um critério de qualidade de serviço, durante um horizonte de previsão limitado. O mesmo modelo foi validado posteriormente por Kusic e Kandasamy (Kusic e Kandasamy, 2007) para otimizar o provisionamento de recurso em ambientes de *Utility Computing*.

Resumindo, a modelagem analítica abrange métodos matematicamente fundamentados e maduros. Todavia, os métodos apresentam várias limitações: modelos analíticos clássicos se baseiam em conhecimento *a priori* da estrutura ou comportamento do sistema, podendo a derivação dos mesmos se tornar difícil para alguns domínios. A dependência de conhecimento *a priori* pode levar também a erros ou suposições irreais, comprometendo a acurácia dos modelos. Finalmente, modelos analíticos são mais difíceis de serem atualizados para se adaptarem a novas condições do sistema. Essas dificuldades motivaram o surgimento da segunda escola ou grupo, a qual abrange os trabalhos que usam técnicas de aprendizado de máquina para prever problemas de desempenho.

O projeto SLIC[2], conduzido nos laboratórios da HP, tem aplicado métodos de aprendizado estatístico para caracterizar problemas de desempenho, incluindo a previsão de problemas. Um produto desse projeto é o estudo apresentado por Powers et al. (Powers et al., 2005), onde métodos tradicionais de regressão e redes Bayesianas são empregados para analisar métricas de sistema e antecipar problemas de degradação de desempenho. O objetivo do estudo é otimizar a atribuição de recursos e o escalonamento de tarefas oportunistas pela antecipação dos períodos de alta ou baixa utilização de recursos. Adicionalmente, o estudo apresenta também uma comparação entre métodos de regressão e métodos de classificação. No entanto, o único método de classificação usado no estudo é a rede Bayesiana.

Classificação também é a técnica escolhida por Andrzejak e Silva (Andrzejak e Silva, 2008) para prever problemas de desempenho causados pelo envelhecimento de aplicações[3]. Os autores descrevem um método para monitoramento de servidores de aplicação baseado em sondas adaptativas e os dados do monitoramento são usados para construir os modelos de envelhecimento. A aplicabilidade das principais famílias de classificadores (incluindo árvore de decisão, redes Bayesiana e *Support Vector Machines*) para construir esses modelos é avaliada. Todavia, o estudo comparativo dos classificadores se restringe

[2]http://www.hpl.hp.com/research/slic/index.html.
[3]Do termo inglês *software aging*.

apenas a avaliar o quão bem os modelos descrevem os dados observados. Outros critérios importantes para a avaliação da adequação dessas técnicas para a área de computação autonômica não são considerados, tais como tempo requerido para treinamento, tempo necessário para classificação, sensibilidade do algoritmo ao tamanho do conjunto de treinamento disponível, etc.

Um estudo sobre técnicas de aprendizado estatístico no contexto de computação autonômica é conduzido por Zhang e Bivens (Zhang e Bivens, 2007). Particularmente, os autores comparam a adequação de redes Bayesianas e redes neurais como ferramentas para modelar o tempo de resposta de sistemas orientados a serviço, contrastando ambas as técnicas através de uma variedade de características relevantes para o campo da computação autonômica e do autogerenciamento. No entanto, o estudo comparativo é limitado pelo fato de considerar apenas as duas técnicas citadas acima e não explorar diferentes parametrizações e configurações dos modelos.

Com relação a mecanismos de previsão de problemas de desempenho, o trabalho descrito nesta tese compartilha alguns pontos em comum, inclusive algumas constatações, com (Powers et al., 2005). A principal diferença é a avaliação de vários métodos de classificação que são confrontados através de uma variedade de características relevantes para o campo da computação autonômica, como será apresentado no Capítulo 4. Dentre essas características, inclui-se o cenário de aprendizado *online* (Capítulo 6). Além disso, é uma preocupação deste trabalho a provisão de mecanismos de previsão robustos. A robustez a falhas transientes dos classificadores tem sido negligenciada pelos trabalhos existentes.

2.4.2 Modelos baseados em Aprendizado Online

Como discutido no Capítulo 1, algoritmos baseados em aprendizado *online* proveem características como entrada de dados continuada, possibilidade de adaptação de modelos em tempo real e baixo consumo de recursos computacionais, que são extremamente desejáveis para um mecanismo de modelagem de comportamento de sistemas computacionais. Esse fato despertou o interesse da área de modelagem de desempenho por algoritmos baseados em aprendizado *online*. De maneira geral, os trabalhos nessa linha podem ser classificados em dois grupos: aprendizado incremental e algoritmos *online*.

Aprendizado incremental refere-se a um conjunto de algoritmos que adaptam técnicas de aprendizado *offline* para serem aplicadas incrementalmente. Cada passo ou incremento do processo de aprendizado faz uso de estruturas de dados eficientes em termos de consumo de recursos computacionais, espe-

cialmente memória e processamento. Um exemplo de trabalho que utiliza essa abordagem é a ferramenta proposta por Kasten e McKinley (Kasten e McKinley, 2007) para construir um módulo de apoio a decisão para aplicações de *stream* de áudio. Esse módulo se baseia em um algoritmo de clusterização no qual o modelo existente é atualizado a cada nova *stream*. Para limitar o consumo de recursos computacionais, padrões similares são agrupados em pequenos grupos denominados esferas sensitivas. Essas proveem uma significativa compressão de dados, além de agilizar os processos de treinamento e classificação. Para manter a acurácia do modelo, as esferas crescem incrementalmente.

Por outro lado, nos algoritmos *online*, o aprendizado assume a forma de uma sequência de tentativas. Em cada tentativa, uma pergunta é apresentada ao módulo de aprendizado, o qual deve fornecer uma resposta. Para responder à pergunta, o módulo de aprendizado usa um mecanismo de previsão que funciona como um mapeamento entre o conjunto de perguntas e o conjunto de respostas admissíveis. Em geral, algoritmos *online* tradicionais conduzem o mapeamento de perguntas e respostas sem o auxílio de uma memória, o que permite uma rápida adaptação a novos conceitos. No entanto, a ausência de experiência passada torna esses algoritmos menos precisos que abordagens incrementais. Schmitt et al. (Schmitt et al., 2008) introduziram um algoritmo *online* padrão, denominado FLORA, em um *framework* de adaptação de contexto para sistemas pervasivos. Para melhorar a eficácia do algoritmo no domínio proposto, os autores propõem algumas modificações, como a adição de memória, a classificação múltipla e a possibilidade de atributos numéricos como valores de entrada.

Considerando aprendizado *online*, este trabalho compartilha com Kasten e McKinley a intensão de avaliar o uso de aprendizado incremental para a construção de modelos de comportamento de sistemas. Todavia, este trabalho difere do anterior de duas formas. Primeiramente, pelas técnicas de aprendizado estatístico usadas, às quais não se inclui clusterização. Segundo, pelo tratamento de mudanças de conceito[4] (Capítulo 6), um problema que pode degradar a acurácia dos mecanismos de previsão quando estes são implantados em ambientes dinâmicos.

2.4.3 Modelos para Diagnóstico de Problemas de Desempenho

Quando sistemas computacionais se comportam mal, seja em virtude de falha ou devido à violação de um objetivo, a compreensão da causa de tal

[4]Do termo inglês *concept drift*.

comportamento pode acelerar a reparação do problema. Nesta seção revisamos alguns trabalhos relevantes na área de diagnóstico de problemas em sistemas autonômicos.

Aguilera et al. (Aguilera et al., 2003), descrevem dois algoritmos para isolar problemas de desempenho em sistemas constituídos por componentes caixa-preta. Um desses algoritmos pode ser aplicado a qualquer estilo de comunicação baseado em troca de mensagem e utiliza técnicas estatísticas de processamento de sinal para inferir relacionamentos de causalidade entre os componentes.

No extremo oposto à abordagem caixa-preta proposta no trabalho anterior, Magpie (Barham et al., 2004) captura, com detalhe, dados sobre a demanda de recursos de uma requisição de serviço, à medida que a requisição é servida. O caminho de cada requisição e os recursos utilizados para atendê-la são gravados em arquivos de *log*. Posteriormente, essas informações são serializadas em *string* de eventos. Para detectar anomalias, as *strings* são analisadas usando técnicas de clusterização. Requisições que não se encontram próximas dos *clusters* existentes são consideradas anômalas. Para detectar o evento ou sequência de eventos que tornam a requisição suspeita, os autores utilizam uma máquina de estado probabilística que modela o processo que gera a sequência de eventos.

Cohen et al. (Cohen et al., 2004) apresentam uma abordagem baseada em redes Bayesianas para correlacionar violações de SLAs (*Service Level Agreements*) e estados do sistema. Os autores avaliam um tipo particular de rede, denominada TAN (*Tree-Augmented Bayesian Network*), para identificar combinações de métricas de sistemas e valores limiares associados a estados de baixo desempenho. TAN impõe uma estrutura de árvore de Markov sobre os dados, sendo portanto uma representação mais compacta do que redes Bayesianas irrestritas. Apesar dessa simplicidade de representação, o modelo conseguiu capturar adequadamente os padrões de desempenho das aplicações testadas. Os resultados apresentados por Cohen et al. motivaram outros trabalhos. Em iManage (Kumar et al., 2007), por exemplo, um agregado de TANs é usado para modelar o comportamento de sistemas de grande escala. Visando uma solução escalável, os autores propõem a divisão do espaço de estado do sistema em partições menores. Para cada partição, um modelo de correlação é construído usando TAN.

Considerando mecanismos de diagnóstico de problemas, nosso trabalho difere dos anteriores na medida que não usamos técnica de aprendizado estatístico para identificar métricas de sistema (ou aplicação) que correlacionam-se com desempenho. Ao contrário, este trabalho faz uso de testes estatísticos

para esse propósito. Diferentemente do que acontece no processo de previsão, onde a resposta do preditor é direta, a investigação das causas de um problema pode exigir uma reflexão sobre os modelos construídos. No entanto, muitas técnicas de aprendizado estatístico não são facilmente interpretadas. Essa constatação pode ser verificada, por exemplo, em Magpie, onde uma máquina de estado probabilística foi modelada para identificar a causa do problema detectado pelo algoritmo de clusterização. Como será mostrado no Capítulo 5, testes estatísticos, ao contrário, são simples e podem ser usados para criar mecanismos de diagnósticos eficientes e eficazes.

2.5 Considerações Finais

Neste capítulo foram introduzidos os fundamentos de gerenciamento de desempenho, sistemas baseados em componentes e computação autonômica. Também foi apresentada uma revisão dos trabalhos mais importantes relacionados com a construção de modelos de comportamento em gerenciamento autonômico de sistemas. Essa revisão levou em consideração trabalhos relacionados com mecanismos de previsão e diagnóstico de desempenho e técnicas aplicadas em cenários de aprendizado *online*. Nesse contexto, foram apresentadas as dificuldades dos trabalhos existentes e as abordagens escolhidas nesta tese para tratá-las. No próximo capítulo, é apresentado o cenário de referência adotado para o desenvolvimento deste trabalho.