PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 49/08

Diagnóstico de Falhas e Localização de Problemas em Sistemas Auto-gerenciáveis

Sand Luz Corrêa

Renato Fontoura de Gusmão Cerqueira

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 49/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Dezembro, 2008

Diagnóstico de Falhas e Localização de Problemas em Sistemas Auto-gerenciáveis[1]

Sand Luz Corrêa e Renato Fontoura de Gusmão Cerqueira

{scorrea, rcerq}@inf.puc-rio.br

Abstract. With the growing size and complexity of computer systems, research in the area of fault diagnosis and problem determination in self-managment systems has started receiving a great deal of attention. The purpose of this work is to provide an overview of the existing research in this area and discuss some limitations of the current approaches in order to fully address the self-management issue, specially in large-scale computing environments.

Keywords: Self-healing, diagnosis, problem determination

Resumo. O crescimento do tamanho e da complexidade dos sistemas computacionais tem despertado a atenção de pesquisas na área de diagnóstico de falhas e determinação de problemas em sistemas auto-gerenciáveis. O objetivo deste trabalho é fornecer uma visão geral da pesquisa nesta área e discutir algumas limitações das abordagens correntes para tratar auto-gerenciamento de forma abrangente, especialmente em ambientes e sistemas de grande escala

Palavras-chave: Auto-cura, diagnóstico, localização de problema

[1]Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil. Processo CNPq número 140729/2006-2.

Responsável por publicações:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introdução

Diagnóstico de falhas e determinação de problemas referem-se, respectivamente, ao processo de detecção de anomalias no comportamento esperado de um sistema e o subseqüente isolamento das causas potenciais que levaram às anomalias observadas. O estudo de problemas de diagnóstico de falhas em sistemas de componentes interconectados remete-se à década de 1960 [1]. Desde então, uma vasta quantidade de trabalhos na literatura têm tratado o problema como um dos grandes desafios ao gerenciamento de sistemas distribuídos. Grande parte destes trabalhos apóiam-se em modelos de dependência, os quais descrevem o relacionamento entre os componentes do sistema.

Modelos de dependência provêem uma forma direta para se identificar possíveis causas de um problema. Dado um componente problemático, o modelo de dependência fornece um caminho, o qual pode ser trilhado para verificar os possíveis candidatos à raiz ou causa principal do problema. Tipicamente, modelos de dependência são construídos a partir da instrumentação de sistemas, que têm seu comportamento monitorado sob ocorrências de falhas específicas ou degradação. Dependências são então reveladas correlacionando-se dados monitorados e trilhando-se a propagação da falha.

Entretanto, à medida que os sistemas crescem em tamanho e complexidade, aumenta-se a dificuldade de se obter, automaticamente, modelos de dependência precisos, detalhados e atualizados. Essa limitação surge porque as abordagens correntes não foram apropriadamente projetadas para tratar questões de imprevisibilidade e dinamismo, características inerentes a sistemas de grande escalas. Sistemas com um grande número de elementos de hardware e software tendem a apresentar um comportamento imprevisível, uma vez que inúmeras combinações de interações entre componentes são permitidas. Como conseqüência, o processo de detecção de anomalia se torna mais desafiador, já que inexiste um comportamento esperado para o sistema. De maneira análoga, um grande número de componentes de sistema propicia uma vasta quantidade de caminhos pelos quais os efeitos de um problema podem se propagar, dificultando o processo de determinação de problemas. Finalmente, o dinamismo dos ambientes de grande escala exige atualizações constantes dos modelos de sistema construídos, uma vez que os mesmos se tornam inconsistentes rapidamente.

Diante deste contexto, este trabalho tem como objetivo fornecer uma visão geral do estado da arte da pesquisa na área de diagnóstico de falhas e determinação de problemas em sistemas auto-gerenciáveis e discutir algumas limitações das abordagens correntes para tratar auto-gerenciamento, especialmente em ambientes e sistemas de grande escala. Para isto, algumas abordagens de solução do problema mais relevantes da literatura são apresentadas, discutindo-se suas principais contribuições. Tais abordagens são apresentadas dentro de uma metodologia de estudo que propõe: *(i)* uma taxonomia para classificar os diversos tipos de abordagens consideradas, comentando-se as vantagens e desvantagens de cada categoria ou classe de soluções; e *(ii)* um conjunto de critérios ou parâmetros que possam ser aplicados no estudo comparativo das soluções abordadas.

Este trabalho está organizado da seguinte forma. Na Seção 2, apresentamos a metodologia adotada que definiu a taxonomia utilizada neste trabalho e os critérios de comparação aplicados no estudo. Na seção 3, descrevemos as abordagens estudadas, organizadas de acordo com a classificação proposta. A Seção 4 traz uma comparação entre as abordagens apresentadas. Finalmente, a Seção 5 apresenta uma discussão sobre a posição atual das abordagens para diagnóstico e localização de problemas em relação a soluções auto-

gerenciáveis inseridas em ambientes computacionais de grande escala.

2 Caracterização da Metodologia Adotada

A metodologia utilizada para elaboração deste trabalho consistiu-se em pesquisa bibliográfica sobre o tema, sendo esta dividida em três partes: levantamento do estado da arte de soluções para diagnóstico de falhas ou degradação em sistemas distribuídos; proposta de uma taxonomia de classificação das diversas abordagens estudas; e definição de um conjunto de critérios, de tal forma que os casos aqui estudados pudessem ser comparados entre si. As abordagens foram escolhidas considerando-se o tipo das mesmas dentro da taxonomia proposta, bem como suas contribuições para a área em questão.

Em relação à taxonomia proposta, para fins deste trabalho, consideramos como diagnóstico de falhas e determinação de problemas os mecanismos usados para detectar anomalias no comportamento de sistemas e, em seguida, isolar as causas principais que conduziram às anomalia observadas. Num contexto mais amplo, utilizamos o termo falha para denotar não somente a causa física ou de algoritmo de um erro [2], mas também para nos referirmos a qualquer problema de desempenho ou degradação observado em um sistema. Neste sentido, as técnicas de diagnóstico de falhas existentes podem ser classificadas em quatro categorias gerais:

- Instrumentação individual de componentes: esta categoria compreende soluções tradicionais para monitoramento de falhas ou desempenho através da instrumentação individual de componentes. Tais soluções são bastante efetivas para detectar anomalias de comportamento em componentes individuais, ou pequenos grupos de componentes. Entretanto, elas falham em fornecer uma visão holística do sistema, uma vez que os comportamentos individuais não são correlacionados com métricas de mais alto nível.

- Correlação com métricas de serviço: ao contrário da categoria anterior, esta classe compreende soluções que correlacionam comportamentos individuais de componentes do sistema com métricas de níveis de serviço, o que garante uma visão mais globalizada do sistema. De maneira geral, estas correlações são implementadas através de aprendizagem entre utilização de recursos e violações de níveis de serviço, ou simplesmente estimadas através de modelos analíticos, como os modelos de fila.

- Soluções baseadas em regras e *enforcement*: esta classe envolve soluções que utilizam políticas baseadas em regras para construir o modelo de comportamento do sistema. Especificação de regras é um método relativamente simples de ser implementado requerendo, entretanto, um conhecimento considerável do domínio do problema. Logo, as abordagens compreendidas nesta classe de soluções são altamente dependentes de aplicações.

- Instrumentação ativa: ao contrário das classes anteriores que se caracterizam pelo monitoramento dos componentes do sistema de forma implícita e passiva, esta classe envolve soluções que se caracterizam pela introdução explicita e sistemática de perturbações nos componentes do sistema e o monitoramento das reações do mesmo em resposta a tais perturbações. Em geral, os resultados dos experimentos de perturbação são usados para alimentar modelos estatísticos que analisam ou estimam a

dependência entre os componentes do sistema. Apesar de bastante direta, esta abordagem possui as seguintes desvantagens: é uma técnica intrusiva e portanto requer muito cuidado ao ser aplicada em sistemas em produção; é uma técnica pouco escalável, pois não é fácil decidir qual componente escolher no processo de perturbação; geralmente exige um grande conhecimento do domínio da aplicação.

Em relação aos critérios de comparação, optamos por um conjunto de requisitos os quais consideramos fundamentais para o processo de caracterização e avaliação de um mecanismo de diagnóstico de falhas. Este conjunto é descrito a seguir:

- Sistema alvo: este critério descreve o tipo de domínio ou sistema para o qual a abordagem foi projetada.
- Tipo de monitoramento: este critério descreve o comportamento do mecanismo de monitoramento em relação ao objeto monitorado. Neste sentido, o mecanismo pode ser classificado como ativo (se introduz explicitamente perturbações no ambiente monitorado) ou passivo (se observa passivamente o ambiente, à medida que os eventos ocorrem normalmente).
- Caracterização de dependência: este critério descreve o tipo de modelo utilizado para caracterizar dependências entre os componentes do sistema. De maneira geral, dois modelos podem ser usados para este fim: modelo baseado em caminhos (*path-based models*) ou baseados em estados (*state-based models*) [3]. Modelos baseados em caminho computam as dependências entre componentes fazendo-se uso dos possíveis caminhos de execução do sistema, enquanto que modelos baseados em estado usam grafos de controle de fluxo, nos quais a transferência de controle entre os diversos estados pode ser descrita por um processo marcoviano.
- Técnica empregada: este critério descreve, de forma resumida, a técnica principal empregada no mecanismo de diagnóstico de falhas.
- Causalidade: este critério mede a efetividade do mecanismo de diagnóstico de falhas ao diferenciar relações de causa (indicando dependência reais) de simples correlações arbitrárias (ou seja, correlações geradas por coincidência).
- Cobertura: este critério mede a efetividade do mecanismo de diagnóstico na cobertura das falhas a que o sistema está exposto.
- Dependência em relação ao domínio: este critério avalia o quanto o mecanismo proposto é dependente de um domínio específico.
- Adaptabilidade: este critério avalia o nível de dificuldade de adaptação do mecanismo proposto em relação a mudanças no comportamento do sistema.

3 Mecanismos de Diagnóstico de Falhas e Determinação de Problemas

3.1 Classe 1: Instrumentação Individual de Componentes

3.1.1 Aguilera et al.

Este trabalho [4] não tem o objetivo de obter ferramentas que automatizam a determinação de problemas em sistemas distribuídos, mas sim desenvolver ferramentas de depuração que facilitem o isolamento de pontos de gargalos de desempenho em sistemas constituídos por componentes caixas-pretas. Neste contexto, a ferramenta apresentada constrói caminhos de causalidades a partir de troca de mensagens entre componentes do sistema. Um caminho representa um conjunto de componentes que estão relacionados causalmente pela troca de mensagens. Cada aresta que conecta um par de componentes num caminho é marcada com o tempo de atraso da chamada. O objetivo da ferramenta é encontrar os padrões de caminhos de causalidade que contribuem mais significativamente para a degradação do sistema. Estes, por sua vez, são os padrões que se repetem com freqüência e que possuem as maiores latências em relação a outros patrões de caminhos igualmente freqüentes. Também é objetivo da ferramenta identificar os componentes participantes de um caminho que contribuem com as maiores latências.

Para tanto, a solução adotada utiliza logs de mensagens trocadas entre nós, durante operação do sistema. Este log é obtido de forma passiva, sem conhecimento prévio de detalhes do domínio da aplicação ou mesmo a semântica de troca de mensagem utilizada. Dois algoritmos são implementados para inferir o caminho de causalidade dominante. O primeiro algoritmo, denominado algoritmo de aninhamento, é utilizado para inferir relações de causalidade entre chamadas que seguem um estilo de comunicação RPC. Para inferir como as chamadas estão aninhadas, o algoritmo examina explicitamente as entradas individuais de um log global de mensagem, verificando a estampa de tempo (*timestamp*) de cada chamada. Desta forma, se um relacionamento de aninhamento aparece repetidamente em um log razoavelmente grande, podemos inferir, com alta probabilidade, que tal aninhamento representa um padrão de relacionamento de causalidade. De forma geral, o algoritmo de aninhamento está dividido em três passos:

1. encontrar pares de chamada (*Call/Return*) no log de mensagens: durante a execução do sistema (monitoramento *on-line*), cada chamada efetuada é registrada no log de mensagens através de uma tupla *(timestamp, type, sender, receiver, id)*, representando, respectivamente, o momento em que a chamada aconteceu, o tipo da chamada (*call* ou *return*), o emissor, o receptor e o identificador da chamada (obtido a partir do cabeçalho do pacote). Durante o processamento do log de mensagens, entradas com o mesmo identificador (id) são combinadas como um par de chamada e retorno. Para cada par de chamada (t_i, s_i, r_i, id_i), identificam-se todos os seus possíveis pares pais $(t_j, \_, s_i, id_j)$ em que $t_j < t_i$.

2. classificar potenciais relações de causalidade: uma vez que o passo anterior pode associar cada par de chamada a múltiplas chamadas pai, é preciso quantificar a potencialidade de todos os relacionamentos elencados. Para isto, um procedimento de pontuação estima o quanto um relacionamento de aninhamento realmente representa um relacionamento de causalidade. Para cada pai potencial (*parent*) de uma

chamada (*child*), calcula-se a diferença de tempo (*delay*) entre a chamada pai e a chamada filha e estima-se o número de vezes em que a tupla *(parent, child, delay)* ocorreu em todo o log. A tupla com maior pontuação representa um relacionamento de causalidade real.

3. Finalmente, no último passo, caminhos de chamadas são derivados a partir dos relacionamentos de causalidade encontrados.

O segundo algoritmo, denominado algoritmo de convolução, pode ser aplicado a qualquer estilo de comunicação baseado em troca de mensagem e utiliza técnicas de processamento de sinal para extrair relacionamentos de causalidade a partir do log de mensagens. Porém, ao contrário do algoritmo de aninhamento, o algoritmo de convolução separa o log em conjuntos por tipo de chamada. Cada conjunto é então tratado como um sinal temporal. Uma chamada de A para B (A, B) é um tipo diferente de uma chamada de B para A (B, A). Essa distinção permite a adequação do algoritmo para diferentes estilos de comunicação. O algoritmo funciona da seguinte forma. Para cada conjunto de mensagens de j para k (j, k) que aconteceram d unidades de tempo após um conjunto de mensagens de i para j (i, j), mensagens V de (i, j) são convertidas para uma função de indicação $s_1(t)$, tal que $s_1(t)$ é igual a 1 se V contem uma mensagem em um intervalo de tempo muito pequeno, envolvendo t. Similarmente, mensagens U de (j, k) são convertidas em uma função de indicação $s_2(t)$. Em seguida, o algoritmo calcula uma função $C(t)$ que estima a correlação entre $s_1(t)$ e $s_2(t)$. $C(t)$ é definida como a função de convolução de s_2 e a inversa no temo de s_1. De maneira geral, a função $C(t)$ possui um pico na posição d, se e somente se, $s_2(t)$ possui uma cópia de $s_1(t)$ deslocada d unidades de tempo.

A grande contribuição deste trabalho consiste no fato dos componentes serem manipulados como caixas-pretas. O termo caixa-preta pode ser aplicado com mais ou menos rigor, dependendo da granulosidade do nó de interesse, o qual pode variar desde uma máquina até um processo ou serviço. Entretanto, quanto menor a granulosidade, menos caixa-preta se torna o componente. Além disso, quando componentes representam máquinas, mensagens trocadas entre serviços no mesmo nó não são consideradas. Isso acarreta medições imprecisas quanto à utilização dos recursos deste nó (CPU, disco, etc). Outra desvantagem deste mecanismo consiste no fato dos dois algoritmos utilizados gerarem uma grande quantidade de falsos positivos. Enquanto tal limitação não representa uma grande ameaça para uma ferramenta de visualização e reportagem de erros, ela pode ser crucial para o sucesso de ferramentas automáticas de diagnóstico de falhas.

3.1.2 Magpie

Magpie [5] é um protótipo, desenvolvida pela Microsoft, com a finalidade de automatizar o diagnóstico e determinação de problemas de desempenho em sistemas voltados para Internet. A ferramenta foi projetada tendo como meta dois objetivos principais. O primeiro consiste na extração de informações sobre o consumo de recursos e fluxo de controle de cada requisição processada. Neste sentido, Magpie provê informações detalhadas sobre como a requisição foi servida e quanto tempo e recursos foram gastos em diferentes pontos do processamento da requisição. O segundo objetivo consiste no uso das informações de requisições individuais para a construção de modelos apropriados para planejamento de carga, depuração de degradação e detecção de anomalias. Para isto, o protótipo foi im-

plementado como um conjunto de ferramentas e seu funcionamento pode ser dividido em três etapas:

1. Instrumentação: O *framework* de instrumentação de Magpie implementa a contabilização da utilização dos recursos do sistema. Essa contabilização e feita por requisição e ocorre à medida que eventos são gerados pelos componentes, tanto no espaço de usuários quanto no modo *kernel*. A atribuição dos eventos às requisições correspondentes se apóia na ordenação dos *timestamp* de criação dos eventos. Cada evento é representado pelo *timestamp* de geração, um nome e um ou mais atributos que o caracterizam. Normalmente, o *framework* de instrumentação de Magpie é implementado pelo *Event Tracing for Windows* (ETW), o qual é capaz de contabilizar consumo de CPU, acesso a disco e envio/recebimento de pacotes de dados em uma máquina. Instrumentação de aplicações e *middlewares* ocorre quando recursos são multiplexados/demultiplexados ou na transferência do fluxo de controle entre componentes.

2. Extração de carga: a instrumentação gera um *stream* alternado de eventos de diferentes requisições, uma vez que estas ocorrem concorrentemente. Como resultado, o primeiro passo na extração da carga é o agrupamento dos eventos de uma mesma requisição para dar início à contabilização. Uma ferramenta denominada *request parser* é usada com este objetivo. Ela identifica os eventos pertencentes a requisições individuais através da aplicação de uma operação de junção (*join*) temporal sobre os mesmo, de acordo com regras especificadas em um esquema de eventos. Durante este processo, a ordenação causal dos eventos não é modificada.

3. Análise de comportamento: Em seguida, os dados obtidos na extração da carga são analisados para construção dos modelos de predição e depuração. Entretanto, antes da análise proceder, os dados são transformados para uma forma canônica, eliminando-se as informações de como a requisições são servidas. A partir da forma canônica, requisições são serializadas de forma determinística para uma representação de *string*, a qual é utilizada na aglomeração (*clustering*). *Strings* similares são agrupadas num mesmo *cluster*, segundo a distância de Levenshtein. *Strings* que não pertencem a nenhum *cluster* suficientemente grande, são consideradas requisições anômalas. Para identificar a causa do problema, Magpie constrói uma máquina de estado probabilística que aceita o conjunto de *strings* representando as requisições possíveis. Requisições anômalas são processadas por esta máquina que identifica todas as transições com baixa probabilidade e os eventos que correspondem a tais transições. Esses eventos são considerados a causa do problema.

Uma grande contribuição de Magpie consiste no *framework* de extração de carga, que relaciona os eventos às requisições sem a necessidade de alocação de identificadores únicos para as requisições.

3.1.3 Pinpoint

Pinpoint é um *framework* muito semelhante a Magpie. Entretanto, ao contrário deste, Pinpoint se destina especificamente à detecção e localização de falhas em aplicações cliente-servidor, não sendo seu objetivo a detecção de problemas de desempenho. O *framework* funciona da seguinte forma:

1. Pinpoint primeiramente grava logs de componentes envolvidos no processamento de requisições. Logs são obtidos a partir de sistemas comerciais que provêem instrumentação no nível do *middleware* e de aplicações. Destes logs, são extraídos dois tipos de comportamento do sistema: caminhos de requisição e interações de componentes. Ambos fornecem visões diferentes do comportamento do sistema. Como em Magpie, um caminho representa uma seqüencia ordenada dos componentes usados para servir a requisição. Entretanto, ao contrário de Magpie, a associação de um evento à requisição é feita através de identificadores únicos. O comportamento anormal dos caminhos é capturados por uma gramática probabilística que calcula a probabilidade de diferentes seqüências terem sido geradas por uma mesma linguagem. Por outro lado, a análise das interações de componentes se apóia na ponderação de cada *link* que representa uma interação entre o componente e outros componentes do sistema. Nesta estratégia, interações anômalas são percebidas comparando-se a contabilização dos *links* que entram e saem de um componente observado, com os valores correspondentes no modelo construído.

2. Em seguida, Pinpoint determina se cada requisição foi completada com sucesso ou falha. Para isto, ele utiliza detectores de falhas externos (como *ping* para detectar queda de máquina) e detectores de falhas internos (que localizam falhas de algoritmo). Como detectores de falhas internas, Pinpoint oferece os dois métodos de determinação de comportamento descritos no item anterior.

3. Finalmente, Pinpoint utiliza duas técnicas independentes para localização de problemas: aglomeração (*clustering*) e árvores de decisão. Ambas as técnicas são utilizadas para encontrar correlações entre a presença de um determinado componente em uma requisição e a falha da mesma.

3.2 Classe 2: Correlação com Métricas de Serviço

3.2.1 Cohen et al.

Este trabalho [6] representa uma das primeiras iniciativas de se utilizar técnicas de aprendizagem estatísticas para capturar a correlação entre dados instrumentados e estados de um sistema, onde este último é expresso através de um conjunto de métricas de qualidade de serviço. Os autores propõem que a correlação de eventos seja formulada como um problema de classificação de padrões. Seja S_t o estado de um SLO (*Service Level Objectives*) no tempo t. S pode assumir um dos estados do conjunto $\{s_+, s_-\}$, representando, respectivamente, conformidade ou violação ao objetivo. Seja M_t o vetor de valores para n métricas $M[M_0,...,M_n]$ coletadas no instante t. A classificação de padrões é então um problema de induzir ou aprender uma função de classificação F, capaz de mapear todos os valores possíveis de M para algum valor de S.

Em seguida, o trabalho propõe que o problema de classificação seja resolvido computando-se o valor de F como a distribuição de probabilidade conjunta representada por uma rede bayesiana sobre $U = \{M_0, ..., M_n, S\}$, onde S é a variável de classificação. A rede bayesiana é induzida estatisticamente a partir de um conjunto de treinamento ou log de observações do sistema em operação, segundo o formato $< M_t, S_t >$.

Comportamentos anômalos são então detectados substituindo-se, na rede bayesiana, os valores observados nas variáveis de M. A rede então calcula o valor de S segundo

sua distribuição de probabilidade. Se $P(s_+|M_t) > P(s_-|M_t)$, então o estado do sistema é normal. Caso contrário, ele é considerado anômalo. Esta relação é também utilizada para se determinar a causa do problema, ou seja, as variáveis monitoradas que contribuem mais para a anomalia detectada. Isto é feito desenvolvendo-se a relação acima, o que resulta na equação 1. Examinando-se a equação, concluí-se que uma variável M_i de M_t está relacionada com a falha detectada se $E_i = log[\frac{P(M_i|M_{pi},s_-)}{P(M_i|M_{pi},s_+)}] > 0$. Adicionalmente, quanto maior o valor de E_i, maior a contribuição de M_i para a anomalia observada.

Uma grande contribuição deste trabalho consiste na estrutura da rede bayesiana utilizada. Os autores optaram por uma NBN (*Naive Bayesian Network* especial, denomina TAN (*Tree-Augmented Bayesian Network*). Esta trata-se de uma rede um pouco mais complexa que uma NBN tradicional, porém muito mais simples que uma rede bayesiana genérica. Os experimentos revelaram que a estrutura escolhida alcança uma boa relação no compromisso entre causalidade e tratabilidade computacional.

$$\sum_i log[\frac{P(M_i|M_{pi},s_-)}{P(M_i|M_{pi},s_+)}] + log\frac{P(s_-)}{P(s_+)} > 0 \quad (1)$$

3.2.2 Zhang et al.

Este trabalho [7] se inspira na proposta de Cohen et al. para resolver problemas de diagnóstico e determinação de problemas de desempenho em sistemas orientados a serviços. Porém, ao contrário do trabalho anterior, os autores destacam a importância de uma abordagem projetada para lidar com dados não observáveis, já que, nestes ambientes, nem todos os serviços provêem dados de desempenho. O trabalho baseia-se em uma rede bayesiana capaz de inferir tempo de resposta do sistema (variável de classificação) a partir de tempos de respostas dos serviços envolvidos em uma requisição (variáveis monitoradas). Para isto, primeiramente a rede bayesiana é usada para derivar tempos de resposta de serviços não observáveis a partir dos serviços observáveis. Em seguida, calcula-se a diferença entre o tempo de resposta real e o tempo de resposta projetado, ou seja o tempo esperado caso os serviços não observados tivessem sido executados normalmente. Quando esta diferença ultrapassa um limiar, o comportamento do sistema é considerado anormal e os serviços responsáveis por tal anormalidade são detectados.

Ao contrário de outras abordagens baseadas em redes bayesianas, é importante ressaltar que, neste trabalho, a distribuição de probabilidade condicional, que descreve o tempo de resposta do sistema, dado os tempos de serviço, não é obtida a partir de treinamento dos dados, mas sim definida deterministicamente através do *workflow* da aplicação. Esta decisão permite que todo o *framework* seja utilizado de forma *on-line*, porém o torna dependente de aplicação.

3.2.3 iManage

iManage [8] consiste num *framework* para diagnóstico e determinação de problemas de desempenho, voltados particularmente para sistemas de grande escala. Para isto, iManage estende o trabalho de Cohen et al., propondo a divisão do espaço de estado do sistema em partes menores e, portanto, mais fáceis de serem gerenciadas. O comportamento das partições é então capturado através da construção de micro-modelos, onde uma rede bayesiana é construída para cada partição.

O algoritmo de particionamento do estado do sistema consiste em uma técnica de aglomeração (*clustering*), onde os estados do sistema representados em cada instância de um mesmo grupo são próximos entre si. Considerando-se S o conjunto de estados do sistema, V o espaço de estado e $V_\alpha \subset V$, o conjunto de variáveis que quando modificadas afetam o estado do sistema, a distância entre dois estados s_1 e s_2 é dado por $v(s_1, s_2) = \eta x \delta_v(s_1, s_2) + \mu x \theta(s_1, s_2)$. μ e η são parâmetros definidos pelo usuário; δ é a distância em relação à dimensão V e θ é a distância em relação a dimensão V_α. Um espaço de estado P deve ser particionado se existem V_α e $V_\alpha^{'}$ tais que:

- $\sum_{\forall s_i, s_j \in S} \delta_{V_\alpha} - V_\alpha^{'}(s_i, s_j) \leq \Delta_{max}$
- $Cardinalidade(V_\alpha^{'}) \leq f$
- f (número máximo de partição) e Δ_{max} (erro máximo permitido) são definidos pelo usuário

Uma vez particionado o espaço de estado, um micro-modelo é construído para cada partição, com o objetivo de inferir os valores de V_α, dados os valores de $V - V_\alpha$. Para isto, uma rede bayesiana é induzida sobre $\{V - V_\alpha, c\}$, onde c, a variável de classificação, representa todos os valores possíveis que a variável V_α pode assumir. Como o espaço em que a rede é construída foi particionado, é possível fazer tal enumeração de forma eficiente.

O particionamento do espaço de estado do sistema é a grande contribuição deste trabalho. Validações feitas pelos autores mostraram que a técnica de partição não somente simplifica a fase de aprendizagem da rede, mas também aumenta a acurácia do modelo representado.

3.2.4 R-Capriccio

R-Capriccio [9] é um *framework* para planejamento da capacidade de sistemas, utilizando-se cargas provenientes de ambientes de produção reais. A partir de tal planejamento, um modelo de utilização de CPU é construído para caracterizar o comportamento normal da CPU em relação à composição de carga do sistema. Anomalias são detectadas comparando-se o comportamento corrente do sistema com o modelo construído. Para isto, R-Capriccio consiste basicamente em três ferramentas: *WorkLoad Profiler*, usada para caracterização da carga do sistema, ou seja os tipos de transações e sessões mais comuns; *Solucionador baseado em Regressão*, ferramenta usada para calcular a demanda de CPU para cada tipo de transação; e um *Modelo analítico*, ferramenta usada para o planejamento efetivo da capacidade do sistema. Estas ferramentas são utilizadas considerando-se as seguintes suposições sobre o sistema:

- O sistema tem uma arquitetura multi-camada
- O gargalo do sistema é a CPU
- Existe uma restrição de tempo de resposta a ser obedecida (um valor limiar de tempo de resposta do sistema)
- Existem logs de aplicações, refletindo todas as requisições e atividades executadas por um cliente.

- É possível mensurar a utilização da CPU em cada camada do sistema.

Para a caracterização da carga, são definidos os conceitos de transação, sessão, capacidade do sistema e tempo de *think*. Uma transação consiste no acesso a uma página Web. Uma sessão consiste num conjunto de transações individuais compondo um serviço. A capacidade do sistema é definida como o número de sessões clientes concorrentes suportadas, sem violar a restrição do tempo de resposta do sistema. O tempo de *think* representa o tempo entre o momento em que o cliente recebe a resposta do servidor e o momento em que ele coloca a próxima requisição. O log contendo os dados da carga é gerado pelo *WorkLoad Profiler*, o qual, em intervalos regulares de tempo, coleta as seguintes informações: a utilização da CPU no período (U^m_{CPU}), o número de transações do tipo i executadas no período (N_i), o número de sessões concorrentes (N) e o tempo médio de *think* (Z).

Em seguida, o *Solucionador* calcula o custo (demanda) de CPU (C_i) para processar cada tipo de transação i, em uma janela de tempo T. Para este cálculo, usa-se um método de regressão estatística, tendo como entrada N_i, T e U^m_{CPU}. Uma vez calculado o valor de cada C_i, é possível calcular o valor da utilização de CPU esperada, ou seja, $U^e_{CPU} = (C_0 + \sum_i N_i.C_i)/T$.

O planejamento da capacidade do sistema utiliza um modelo analítico de teoria de filas. Neste modelo, um sistema multi-camada é normalmente modelado como um sistema fechado (o número de clientes é constante). Cada camada é modelada como um conjunto de pares *(servidor, fila)* e a carga é balanceada entre os recursos de uma mesma camada. O tempo de *think* é modelado como um servidor infinito. Uma vez calculado o tempo de serviço de cada fila, em função de C_i, é possível resolver o sistema usando-se MVA (*Mean-Value Analysis*). Entretanto, MVA supõe que o tempo de serviço em cada fila é constante. Para resolver este problema, aplica-se MVA em cada janela de tempo T e, posteriormente, os resultados obtidos são combinados entre si.

3.3 Classe 3: Soluções baseadas em regras

3.3.1 Bhat et al.

Neste trabalho [10] é apresenta uma extensão para o Accord [11] (um *framework* para construção de aplicações autônomas), onde a estrutura de auto-gerenciamento das aplicações é complementada com modelos baseados em teoria de controle e estratégias de otimização.

Em Accord, o comportamento dos elementos de uma aplicação e suas interações podem ser gerenciados em tempo de execução usando-se um conjunto de regras definidas dinamicamente. Para isto, cada serviço autonômico é monitorado e controlado por um serviço de gerenciamento através de uma porta de controle, a qual exporta o comportamento do serviço monitorado. O fluxo de controle de uma aplicação é decomposto em regras (de comportamento e de interações), cujo alvo são serviços individuais. Estas regras são então direcionadas para os respectivos gerenciadores de cada serviço, onde são executadas para adaptar o comportamento dos serviços gerenciados. Regras são representadas como estruturas do tipo *condition-then-action*.

Na extensão proposta, modelos de controle ou LLC (*limited look-ahead control*) são adicionados aos gerenciadores de serviço, complementando-se as estratégias baseadas em regras. Estes controladores, construídos a partir de modelos matemáticos formais, são inseridos para aumentar a efetividade das regras definidas, já que estas são suscetíveis a erros. A abordagem por LLC permite que múltiplos objetivos (QoS) e restrições do

sistema sejam representadas explicitamente no problema de otimização, em cada passo de controle. Dado um passo de controle k, o controlador encontra o menor valor que minimiza a função de custo $\sum_{i=k+1}^{k+N} J(x(i), u(i))$, sujeita às restrições do sistema. N representa um horizonte de previsão, $x(i)$ representa o estado do sistema no passo k e $u(i)$ representa as variáveis de controle e parâmetros do ambiente no tempo k.

Outro trabalho interessantes envolvendo diagnóstico e determinação de problemas através de especificação de regras pode ser encontrados em [12]. Ele descreve uma abordagem baseada em sistemas multi-agentes para extrair e analisar informações de contexto, as quais são posteriormente utilizadas para auto-diagnóstico de falhas.

3.4 Classe 4: Instrumentação Ativa

3.4.1 Active Dependency Discovery

Active Dependency Discovery (ADD) [13] é uma abordagem para determinação de problemas em que a descoberta de dependência entre os componentes do sistema ocorre de forma ativa, através de perturbações no ambiente de execução. Para tanto, inicialmente, os recursos e serviços principais do sistema são identificados. Em seguida, alguns recursos são escolhidos como alvo para perturbação. A reação do sistema em resposta à perturbação nos componentes selecionados é capturada em um conjunto de dados monitorados. Análises de regressão estatística nestes dados possibilitam a indicação da presença, bem como a quantificação do grau de dependência entre os outros componentes do sistema e os componentes perturbados. Tais informações são então usadas para construir um grafo de dependência entre os componentes do sistema.

Um grande problema da aplicação de ADD para determinação de problemas, além daqueles inerentes à abordagens de instrumentação ativa, é sua baixa cobertura. Isto se deve porque o processo de descoberta de dependência dependente da perturbação aplicada. Esta, por sua vez, depende da escolha apropriada de um conjunto de falhas que serão injetadas em um conjunto de componentes. Para resolver este problema, [14] sugere uma solução para identificar falhas apropriadas, ou seja, falhas que, quando injetadas no ambiente, conduzem a um processo de descoberta de dependência entre seus componentes.

Para tanto, a solução proposta se baseia em um modelo hierárquico de falhas. Nesta técnica, os efeitos de falhas em componentes de baixo nível são propagadas para componentes de alto nível através de um dicionário de falhas. O dicionário se torna um catálogo que detalha falhas em componentes mais simples do sistema (baixo nível) e os respectivos impactos que as mesmas causam em componentes mais complexos (alto nível). Desta forma, conhecendo-se uma falha de baixo nível, sua localização e alguma informação temporal, é possível inspecionar a entrada correspondente no dicionário de dados e encontrar as falhas de alto nível causalmente relacionadas. Uma vez construído o dicionário e conhecendo-se as falhas de baixo nível de um ambiente, é possível desenvolver e implantar injetores que inserem falhas de alto nível. Portanto, esta abordagem considera que falhas de baixo nível são geralmente mais fáceis de serem caracterizadas, enquanto que injetores de falhas podem ser acelerados se a injeção é executada usando-se modelos de falhas de alto nível.

3.4.2 Brodie et al.

Neste trabalho [15], os autores tratam o problema de diagnóstico de falhas através do uso de *probes*. A idéia básica do trabalho consiste no uso de uma abordagem mais ativa para o

problema de diagnóstico, fornecendo-se um compromisso entre efetividade de uma técnica mais direta e o seu custo de implementação. Um *probe* consiste num comando enviado por uma máquina particular (denominada *probing station*) para um servidor, a fim de testar um serviço específico. Como resultado, o *probe* retorna um conjunto de medições, como tempo de resposta, código de status, etc. Entretanto, o uso de *probes* pode gerar uma sobrecarga excessiva no sistema. Para tornar seu custo de implantação mais atrativo, os autores propõem: *(i)* uma fase de planejamento, na qual um conjunto pequeno, porém efetivo, de *probes* serão selecionados; *(ii)* uma fase de diagnóstico, onde o processo de determinação do problema é executado usando-se os resultados dos *probes*.

O problema da seleção de *probes* é resolvido encontrando-se o menor subconjunto de *probes* capaz de diagnosticar o maior número possível de problemas. A seleção de um conjunto de *probes* P é feita através do conceito de capacidade de diagnóstico do conjunto, denominada $H(P)$. Esta é definida como a entropia condicional $H(N|G)$, onde $N = \{1, ..., N\}$ denota o nó e $G = \{1, ..., k\}$ denota qual grupo contem o nó na decomposição induzida por P. Desta forma, dois algoritmos podem ser utilizados para encontrar o conjunto mínimo. O primeiro algoritmo, denominado busca subtrativa, inicia-se com o conjunto de todos os *probes* disponíveis. Cada *probe* é então considerado, sendo descartado se a capacidade de diagnóstico do conjunto continuar a mesma depois de sua remoção. O segundo algoritmo consiste em uma estratégia gulosa. Ela inicia-se com um conjunto vazio de *probes* e, a cada etapa, o *probe* que fornecer a decomposição com maior capacidade de diagnóstico é adicionado ao conjunto.

Uma vez selecionado o conjunto de *probes*, ele é utilizado para diagnóstico de falhas. Isto é feito através de uma rede bayesiana que incorpora dependências probabilísticas entre possíveis falhas na rede (causas) e os resultados obtidos a partir do conjunto de *probes* (sintomas).

4 Estudo comparativo

Nesta seção, apresentamos uma comparação entre as diversas abordagens apresentadas, avaliando-as segundo os critérios definidos na Seção 2. As tabelas 1, 2 e 3 resumem o estudo comparativo, descrito abaixo:

- Sistema alvo: Aguilera et al. destina-se a qualquer aplicação distribuída que se comunica por um mecanismo de troca de mensagem; Magpie, Pinpoint, Cohen et al., iManage, R-Capriccio, ADD e Brodie et al. se aplicam principalmente a sistemas cliente-servidor para Internet; Zhang et al. se destina a arquiteturas orientadas a serviço e Bhat et al. se destina especificamente a aplicações baseadas em componentes Accord.

- Tipo de monitoramento: excetuando-se ADD e Brodie et al., todas as demais abordagens estudas utilizam monitoramento passivo.

- Modelo de caracterização de dependência: as soluções que rastreiam requisições, quebrando-as em caminhos de execução, implementam a abordagem *path-based*. Este é o caso de Aguilera et al., Magpie e Pinpoint. Por outro lado, Cohen et al., Zhang et al., iManage e Brodie et al. constroem modelos de dependência através de redes bayesianas. Como estas incorporam a suposição de Markov (cada variável X_i é

independente do seus não descendentes, dado seu pai) estas soluções se baseiam numa abordagem *state-based* em relação ao modelo de dependência. Bhat et al. também utiliza um modelo de dependência *state-based*, uma vez que definições de regras do tipo *event-condition-action* podem ser descritas por máquinas de estado. Contrária às abordagens anteriores, ADD se baseiam em um modelo de dependência hierárquico, onde falhas de alto nível são decompostas em falhas de baixo nível.

- Causalidade: como mencionado anteriormente, o mecanismo de construção de caminhos adotada em Aguilera et al. gera muitas relações aninhadas que não expressam nenhuma relação de causalidade. De forma semelhante, em ADD, o processo de escolha dos elementos a serem perturbados não é automática, resultando em uma solução para levantamento de causalidade pouco efetiva. Nas demais abordagens, os mecanismos de construção de modelos de dependência capturam efetivamente a causalidade existente entre os componentes. Dentre tais abordagens, porém, a solução de particionamento de espaço proposta em iManage garante modelos mais precisos.

- Cobertura: de maneira geral, a cobertura das abordagens que utilizam métodos de aprendizagem estatística (Aguilera et al., Magpie, Pinpoint, Cohen et al., iManage, R-Capriccio e Brodie et al.) depende da amostra utilizada no processo de treinamento do modelo. Entretanto, em Zhang et al., a cobertura está diretamente relacionada com a precisão do *workflow* da aplicação, já que este substitui o processo de treinamento do modelo. Por outro lado, Bhat et al. e ADD se apóiam em mecanismos que dependem de intervenções humanas para sua concretização e, portanto, apresentam baixa cobertura.

- Dependência em relação a um domínio: Magpie depende da especificação do esquema de eventos para relacioná-los a requisições; Zhang et al. depende que o desenvolvedor forneça o *workflow* da aplicação; a especificação das regras que governam o comportamento de uma aplicação em Bhat et al. é totalmente dependente da aplicação; o processo de perturbação do ambiente usado em ADD exige o conhecimento sobre o domínio específico. Nas demais abordagens, o mecanismo de diagnóstico e localização de falhas opera de forma independente a qualquer domínio específico.

- Adaptabilidade: de maneira geral, as abordagens estudadas apresentam pouca capacidade de adaptação. Este é o caso, por exemplo, das soluções baseadas em métodos de aprendizagem estatística, as quais apresentam grande dificuldades de atualização dos modelos construídos. Por outro lado, soluções baseadas em regras podem apresentar maior capacidade de adaptação, quando aplicadas em ambientes menores e mais simples.

Abordagem	**Monitoramento**	**Caracterização de Dependência**	**Técnica Empregada**
Aguilera et al.	passiva	*path-based*	pontuação de caminhos e processamento de sinal
Magpie	passiva	*path-based*	*clustering*
Pinpoint	passiva	*path-based*	gramática probabilística ou *clustering* ou árvore de decisão
Cohen et al.	passiva	*state-based*	redes bayesianas
Zhang et al.	passiva	*state-based*	redes bayesianas
iManage	passiva	*state-based*	multi-modelos de redes bayesianas
R-Capriccio	passiva	*state-based*	regressão estatística e teoria de filas
Bhat et al.	passiva	*state-based*	regras e teoria de controle
ADD	ativa	hierárquica	dicionário de falhas
Brodie et al.	ativa	*state-based*	probes e redes bayesianas

Tabela 1: Estudo Comparativo: Comparação das abordagens estudas em relação ao tipo de monitoramento, caracterização de dependência a técnicas empregadas

Abordagem	**Causalidade**	**Cobertura**	**Sistemas Alvo**
Aguilera et al.	baixa	dependente da amostra	troca de mensagem
Magpie	alta	dependente da amostra	Internet
Pinpoint	alta	dependente da amostra	Internet
Cohen et al.	alta	dependente da amostra	Internet
Zhang et al.	alta	dependente do workflow da aplicação	orientado a serviço
iManage	muito alta	dependente da amostra	Internet
R-Capriccio	alta	dependente da amostra	Internet
Bhat et al.	alta	baixa	aplicações Accord
ADD	baixa	baixa	Internet
Brodie et al.	alta	dependente da amostra	Internet

Tabela 2: Estudo Comparativo: Comparação das abordagens estudas em relação a causalidade, cobertura e sistemas alvo

Abordagem	**Domínio**	**Adaptabilidade**
Aguilera et al.	independente	baixa
Magpie	dependente	baixa
Pinpoint	independente	baixa
Cohen et al.	independente	baixa
Zhang et al.	dependente	baixa
iManage	independente	baixa
R-Capriccio	independente	baixa
Bhat et al.	dependente	média
ADD	dependente	baixa
Brodie et al.	independente	baixa

Tabela 3: Estudo Comparativo: Comparação das abordagens estudas em relação a dependência do domínio de aplicação e adaptabilidade

5 Conclusão

Este trabalho apresentou uma visão geral do estado-da-arte da pesquisa na área de diagnóstico de falhas e determinação de problemas em sistemas auto-gerenciáveis. Em geral, o estudo nos permitiu identificar os trabalhos mais relevantes nesta área, avaliar seus mecanismos de diagnóstico de falhas e investigar a posição dos mecanismos atuais em relação à questão do auto-gerenciamento. Este estudo nos permitiu as seguintes conclusões:

1. Os modelos construídos ainda não conseguem capturar com abrangência as várias características e aspectos comumente presentes em sistemas complexos.

2. Como conseqüência, as soluções apresentadas ainda são protótipos imaturos que tratam isoladamente alguns aspectos relevantes para o auto-gerenciamento.

3. A grande maioria das soluções não estão efetivamente preparadas para tratar adaptabilidade. Os modelos construídos não são facilmente atualizados e, esta atualização dificilmente pode ser feita de forma *on-line*.

4. Os mecanismos de inferência ainda são limitados, complexos e de difícil parametrização

5. Grande parte das soluções não foram validadas em ambientes computacionais de grande escala, dificultando a avaliação da escalabilidade dos mecanismos.

Referências

[1] BIRNBAUM, L. W.. **On the Importance of Different Components in a Multi-Component System**. In: Krishnaiah, P. R., editor, MULTIVARIATE ANALYSIS II, p. 581–592. Academic Press, 1969.

[2] AVIZIENIS, A.; LAPRIE, J. C.; ; RANDELL, B.. **Dependability and Its Threats: A Taxonomy**. In: PROCEEDINGS OF IFIP CONGRESS TOPICAL SESSIONS, p. 91–120, 2004.

[3] GOKHALE, S.. **Architecture-Based Software Reability Analysis: Overview and Limitations**. Transactions on Dependable and Secure Computing, 4(1):32–40, 2007.

[4] AGUILERA, M.; MOGUL, J.; WIENER, J.; REYNOLDS, P. ; MUTHITACHAROEN, A.. **Performance Debuggin for Distributed Systems of Black Boxes**. In: PROCEEDINGS OF THE NINETEENTH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, p. 74–89, 2003.

[5] BARHAM, P.; DONNELLY, A.; ISAACS, R. ; MORTIER, R.. **Using Magpie for Request Extraction and Workload Modelling**. In: PROCEEDINGS OF THE 6TH CONFERENCE ON SYMPOSIUM ON OPEARTING SYSTEMS DESIGN AND IMPLEMENTATION, 2004.

[6] COHEN, I.; GOLDSZMIDT, M.; KELLY, T.; SYMONS, J. ; CHASE, J.. **Correlating Instrumentation Data to System States: a Building Block for Automated Diagnosis and Control**. In: PROCEEDINGS OF THE 6TH CONFERENCE ON SYMPOSIUM ON OPEARTING SYSTEMS DESIGN AND IMPLEMENTATION, p. 231–244, 2004.

[7] ZHANG, R.; MOYLE, S.; MCKEEVER, S. ; BIVENS, A.. **Performance Problem Localization in Self-healing, Service-oriented Systems using Bayesian Networks**. In: PROCEEDINGS OF THE 2007 ACM SYMPOSIUM ON APPLIED COMPUTING, p. 104–109, 2007.

[8] KUMAR, V.; COOPER, B.; EISENHAUER, G. ; SCHWAN, K.. **iManage: Policy-Driven Self-Management for Enterprise-Scale System**. In: PROCEEDINGS OF THE 8TH INTERNATIONAL MIDDLEWARE CONFERENCE, 2007.

[9] ZHANG, Q.; CHERKASOVA, L.; MATHEWES, G.; GREENE, W. ; SMIRNI, E.. **R-Capriccio: A Capacity Planning and Anomaly Detection Tool for Enterprise Services with Live Workloads**. In: PROCEEDINGS OF THE 8TH INTERNATIONAL MIDDLEWARE CONFERENCE, 2007.

[10] BHAT, V.; PARASHAR, M.; LIU, H.; KHANDEKAR, M.; KANDASAMY, N. ; ABDELWAHED, S.. **Enabling Self-Managing Applications using Model-based Online Control Strategies**. In: PROCEEDINGS OF THE THIRD IEEE INTERNATIONAL CONFERENCE ON AUTONOMIC COMPUTING (ICAC'06), p. 15–24, 2006.

[11] LIU, H.; PARASHAR, M.. **A Component based Programming Framework for Autonomic Applications**. In: PROCEEDINGS OF THE 1ST INTERNATIONAL CONFERENCE ON AUTONOMIC COMPUTING, p. 10–17, 2004.

[12] PARK, J.; YOO, G. ; LEE, E.. **Proactive Self-Healing System based on Multi-Agent Technologies**. In: PROCEEDINGS OF THE THIRD ACIS INT'L CONFERENCE ON SOFTWARE ENGINEERING RESEARCH, MANAGEMENT AND APPLICATIONS, p. 256 – 263, 2005.

[13] BROWN, A.; KAR, G. ; KELLER, A.. **An Active Approach to Characterizing Dynamic Dependency for Problem Determination in a Distributed Environment**. In: 7TH IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGMENT, p. 377–390, 2001.

[14] BAGCHI, S.; KAR, G. ; HELLERSTEIN, J.. **Dependency Analysis in Distributed System using Fault Injection: Application to Problem Determination in an e-commerce Environment**. In: 12TH INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEM: OPERATIONS AND MANAGEMNT, 2001.

[15] BRODIE, M.; RISH, I. ; MA, S.. **Intelligent Probing: A Cost-Efficient Approach to Fault Diagnosis in Computer Networks**. IBM Systems Journal, 41(3):372–385, 2002.