Nemo: Procurando e Encontrando Anomalias em Aplicações Distribuídas

Brivaldo A. da Silva Junior[1], **Fabrício B. de Carvalho**[1], **Ronaldo A. Ferreira**[1]

[1]Faculdade de Computação (FACOM)
Universidade Federal de Mato Grosso do Sul (UFMS)
CEP – 79070-900 – Campo Grande – MS – Brasil

{brivaldo,fabricio,raf}@facom.ufms.br

Abstract. *Diagnosing anomalies in large enterprise networks consumes significant time of technical support teams, mainly because of the numerous complex interactions among the applications and network elements (servers, routers, links, etc.). The most complete and promising approach for solving this problem, called Sherlock, uses network traces to automatically build an Inference Graph (IG) that models the multiple interactions and dependencies present in a distributed environment. Despite the progress provided by Sherlock in the problem modeling, its execution time for inferring the probable causes and the precision of its anomaly detection results leave much room for improvements. This work proposes Nemo, a tool that explores domain-specific knowledge and a theoretical property of Bayesian Networks to significantly reduce the IG and consequently the execution time. Simulation results using real and synthetic data show that Nemo reduces Sherlock's execution time by over 90% and improves its precision in all simulated scenarios.*

Resumo. *Diagnosticar anomalias em grandes redes corporativas consome tempo considerável das equipes de suporte técnico, principalmente pela complexidade das inúmeras interações existentes entre as aplicações e os elementos de rede (servidores, roteadores, enlaces, etc.). A abordagem mais completa e promissora para a solução desse problema, denominada Sherlock, utiliza traços de rede para construir automaticamente um Grafo de Inferência (GI) que modela as múltiplas interações e dependências presentes em um ambiente distribuído. Apesar do progresso feito por Sherlock na modelagem do problema, o seu tempo de execução para se inferir as possíveis causas e a precisão dos resultados de detecção das anomalias ainda deixam a desejar. Este trabalho propõe Nemo, uma ferramenta que explora conhecimento específico do domínio do problema e uma propriedade teórica de Redes Bayesianas para reduzir significativamente um GI e, consequentemente, o tempo de execução. Resultados de simulação utilizando dados reais e sintéticos mostram que Nemo reduz o tempo de execução de Sherlock em mais de 90% e melhora sua precisão em todos os cenários simulados.*

1. Introdução

As grandes corporações dependem cada vez mais da Internet para desenvolver suas atividades cotidianas. Elas utilizam a grande rede para vender produtos, para troca de mensagens entre funcionários e parceiros, ou para divulgar suas marcas e serviços. A proliferação de atividades desenvolvidas na Internet torna necessária a construção de ambientes distribuídos mais sofisticados para dar vazão a essas novas demandas, além de levar a aplicações distribuídas

mais complexas e dependentes de vários componentes da infraestrutura de comunicação. Uma aplicação Web, por exemplo, pode ser executada em um cluster com servidores interligados por inúmeros roteadores em um *Data Center*, interagir com um servidor de banco de dados e fazer consultas a vários serviços distribuídos, como *Domain Name System* (DNS), *Active Directory* (AD), *Windows Internet Name Service* (WINS), *Radius*, etc. Consequentemente, anomalias, provenientes de falhas ou sobrecargas, em um dos componentes desse novo ambiente podem levar ao mal funcionamento de uma aplicação e uma percepção ruim dos clientes.

Nesse novo cenário, detectar anomalias em ambientes distribuídos tem se tornado uma tarefa extremamente desafiadora. Ferramentas tradicionais de gerenciamento, como Openview [HP 1990], Tivoli [IBM 1996], nmap [Lyon 1997], Nagios [Galstad 1999], ping e traceroute, deixam muito a desejar, pois fornecem informações de alta granularidade e dependem demasiadamente das habilidades dos administradores de rede para detecção e diagnóstico de problemas. Essas ferramentas, normalmente, fornecem apenas informações estatísticas, de conectividade ou de disponibilidade de um componente de software ou de hardware da rede, mas são incapazes de determinar os relacionamentos e as dependências entre eles, dificultando o rápido diagnóstico de um problema observado pelos usuários da rede.

As principais propostas para detecção e diagnóstico de anomalias em ambientes distribuídos existentes na literatura podem ser divididas em dois grandes grupos: intrusivas e não intrusivas. Nas propostas intrusivas, os principais exemplos são Magpie [Barham et al. 2004], Pinpoint [Chen et al. 2004], PIP [Reynolds et al. 2006] e vPath [Tak et al. 2009]. Nessas propostas, as aplicações são modificadas e instrumentadas para registrar os caminhos causais, ou seja, os caminhos percorridos por uma determinada requisição. Com os caminhos causais, é possível determinar os componentes de software e hardware envolvidos no processamento de uma requisição. Dessa forma, uma anomalia em uma aplicação pode ser mais facilmente diagnosticada analisando-se somente o conjunto de componentes utilizados no processamento da requisição. Algumas propostas, como o X-Trace [Fonseca et al. 2007], não se limitam a apenas instrumentar as aplicações, mas toda a pilha de protocolos, aumentando a visibilidade de execução das aplicações.

Muito embora as abordagens intrusivas produzam resultados mais confiáveis, elas dependem fortemente de programadores para instrumentar as aplicações, além de não serem úteis para monitorar aplicações legadas. Algumas abordagens sugerem a instrumentação de *frameworks* para essas situações [Barham et al. 2004], mas nesses casos a efetividade do diagnóstico cai bastante, além de não cobrir as aplicações legadas que não utilizam *frameworks*.

Os principais exemplos de abordagens não intrusivas Sherlock [Bahl et al. 2007], Constellation [Barham et al. 2008], Spotlight [John et al. 2010] e NSDMiner [Natarajan et al. 2012] não modificam as aplicações ou *frameworks* e utilizam somente informações de traços de rede para inferir as causas dos problemas. As principais diferenças de cada abordagem são determinadas, basicamente, pelas informações analisadas, pelas metodologias estatísticas de inferência e pelas soluções sistêmicas dos problemas inerentes à coleta de informações.

Embora as abordagens não intrusivas sejam menos precisas que as intrusivas, elas são consideradas mais viáveis na prática, pois não exigem conhecimento prévio das aplicações e nem modificações para incluir mecanismos de registro de eventos, além de serem úteis e apropriadas para ambientes legados e pouco documentados.

A abordagem não intrusiva mais completa e promissora para detecção e diagnóstico de anomalias em ambientes distribuídos, denominada Sherlock, utiliza traços de rede para construir automaticamente um Grafo de Inferência (GI) que modela as múltiplas interações e dependências presentes em um ambiente distribuído. As dependências são probabilísticas e representadas por pesos (probabilidades) nas arestas que interligam os nós (serviços, roteadores, enlaces, etc.). Apesar do progresso feito por Sherlock na modelagem do problema, o seu tempo de execução para se inferir as possíveis causas e a precisão dos resultados de detecção das anomalias ainda deixam a desejar. Sherlock utiliza um algoritmo de inferência Bayesiana (Ferret) bastante custoso cuja função de pontuação inclui todos os nós do grafo, tornando-o não escalável em cenários em que o GI possui milhares de nós.

Este trabalho propõe Nemo (*Network Monitor*), uma ferramenta não intrusiva que explora conhecimento específico do domínio do problema e uma propriedade teórica de Redes Bayesianas para reduzir significativamente um GI e, consequentemente, o tempo de execução. Resultados de simulação utilizando dados reais e sintéticos mostram que Nemo reduz o tempo de execução de Sherlock em mais de 90% e melhora sua precisão em todos os cenários simulados.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 descreve a ferramenta Nemo, seus algoritmos e suas propriedades teóricas. A Seção 3 descreve brevemente aspectos de implementação da ferramenta. A Seção 4 apresenta os resultados dos experimentos realizados e a Seção 5 conclui este trabalho.

2. Ferramenta Nemo

A ferramenta Nemo foi inspirada no Sherlock e utiliza boa parte de sua modelagem conceitual, bem como parte das estruturas de dados que foram inicialmente introduzidas em [Bahl et al. 2007]. Entretanto, Nemo introduz novos conceitos teóricos e algoritmos bem mais eficientes que os utilizados por Sherlock. Assim como Sherlock, a ferramenta Nemo é dividida em dois componentes principais: agente e gerente. A organização geral de Nemo está ilustrada na Figura 1.

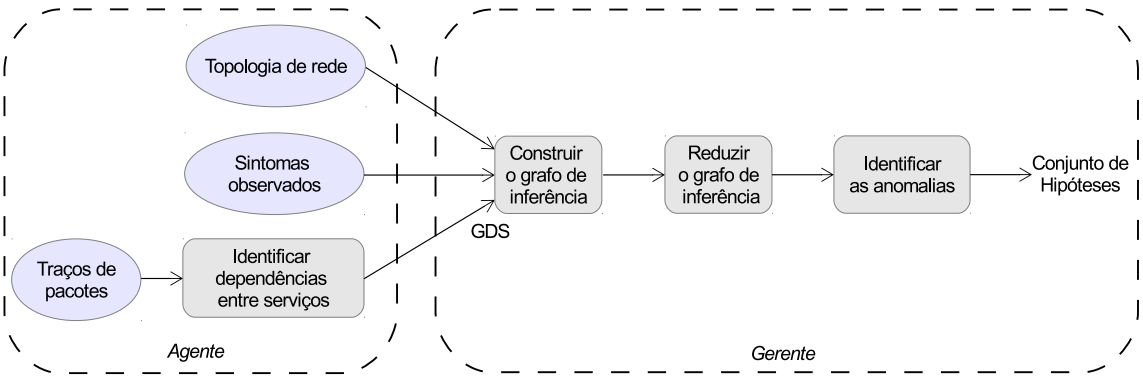

Figura 1. Visão geral da ferramenta Nemo.

Os agentes de Nemo são instalados em estações de trabalho e monitoram os pacotes que são enviados e recebidos pelas estações. A partir dos pacotes monitorados, um agente constrói

localmente um Grafo de Dependências de Serviços (GDS) para os serviços acessados pela estação e monitora os tempos de resposta de cada acesso para construir estatísticas confiáveis sobre o tempo médio de acesso de cada serviço. Quando o tempo de resposta é muito superior ao tempo médio, indicando possivelmente uma anomalia, os agentes solicitam ao gerente que seja executado o processo de diagnóstico de anomalias. Além disso, os agentes coletam informações de topologia da rede, por meio de um módulo de traceroute implementado internamente nos agentes, e enviam essas informações para o gerente.

O gerente é responsável por combinar todas as dependências entre serviços fornecidas pelos agentes e consolidar as visões locais em um GDS global para toda a rede. Além disso, o gerente faz a união do GDS com as informações de topologia para gerar um Grafo de Inferência (GI). O GI captura todas as dependências entre serviços e elementos de rede como enlaces, roteadores e servidores. O gerente também executa o processo de diagnóstico de anomalias quando solicitado pelos agentes. A Seção 2.1 descreve mais detalhadamente como os GDSs são construídos por Nemo.

As principais diferenças entre Nemo e Sherlock estão nos módulos de redução do GI (inexistente em Sherlock) e no módulo de identificação de anomalias. Muito embora a ferramenta tenha exigido um grande esforço de implementação e implantação em um ambiente de produção, o foco deste trabalho, devido à limitação de espaço, será nas melhorias teóricas e algorítmicas introduzidas nos módulos de redução do GI e no de detecção de anomalias. Esses módulos serão discutidos nas Seções 2.4 e 2.5, respectivamente. Os aspectos mais relevantes da implementação da ferramenta serão discutidos na Seção 3.

As próximas seções introduzem conceitos teóricos e descrevem as principais estruturas de dados utilizadas neste trabalho.

2.1. Grafo de Dependências de Serviços

O Grafo de Dependências de Serviços (GDS) descreve as dependências existentes entre serviços. Cada serviço acessado por um cliente, e capturado pelos agentes de Nemo, é mapeado para um nó do GDS. Um serviço S_1 é dependente de um serviço S_2 se o acesso a S_1 é normalmente precedido por um acesso a S_2. O GDS representa essa dependência por uma aresta direcionada de S_2 para S_1 e atribui um peso (probabilidade) a essa aresta de acordo com a força da dependência. A probabilidade de dependência (peso da aresta) é calculada dividindo-se o número de vezes que acessos ao serviço S_2 precedem acessos a S_1 dentro de um intervalo limite de tempo (intervalo de dependência) pelo número de vezes que S_1 é acessado em todos os traços coletados pelos agentes. O intervalo de dependência é um parâmetro do sistema e pode ser alterado de acordo com características específicas do sistema sendo monitorado. O valor sugerido por Sherlock e também utilizado neste trabalho é fixo e igual a 10ms. Esse valor se mostrou bastante eficaz na captura das dependências.

A Figura 2 ilustra o mapeamento da dependência entre os serviços HTTP e DNS. Nesse caso, como o acesso ao serviço HTTP ocorre dentro do intervalo de dependência do acesso ao serviço DNS, infere-se que o serviço HTTP é dependente do serviço DNS. A probabilidade da dependência é calculada de acordo com o número de vezes que esses dois acessos ocorrem dentro do intervalo de dependência. Observe que essa maneira de se calcular as probabilidades de dependência captura adequadamente situações em que as respostas de DNS são armazenadas em cache e que acessos seguidos ao mesmo endereço não precisam de novos acessos ao servidor de nomes. Nesse caso, a probabilidade de dependência será menor que 1,0.

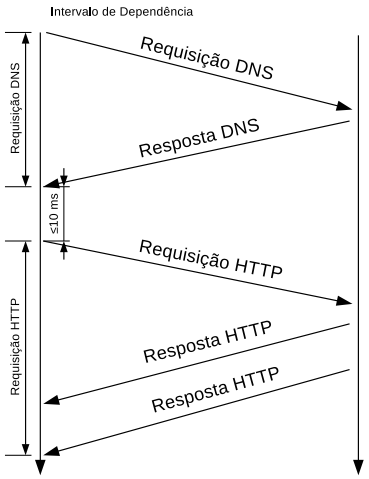

Figura 2. Mapeamento de dependências entre serviços.

2.2. Grafo de Inferência de Rede

Um GDS captura adequadamente as dependências entre serviços, mas não inclui informações sobre a infraestrutura de comunicação. Um Grafo de Inferência (GI) estende um GDS e é definido como um Grafo Direcionado Acíclico (GDA), com pesos nas arestas, que representa as dependências entre componentes de um ambiente distribuído, sendo que componentes incluem serviços e elementos de hardware. Em um GI, há três tipos de nós:

- **nós de observação:** modelam a experiência que o usuário está observando ao acessar um serviço. Essa experiência representa o tempo de acesso ao serviço e pode indicar que o serviço está *ativo* (operando normalmente e com tempos de resposta baixos), *problemático* (operando, mas com tempos de resposta muito altos), ou *inativo*.
- **nós raízes de problema:** representam servidores, equipamentos de rede, enlaces ou serviços. Quando um nó desse tipo apresenta um problema, alguns nós de observação acusarão o problema, fazendo com que o usuário perceba a degradação de um serviço.
- **meta nós ou nós de interligação:** servem para interligar os nós de observação a outros nós do GI que podem ser causadores de problemas. Eles são de três tipos: ruído máximo (ligações um-para-um), seleção (modelam balanceadores de carga) e *failover* (modelam redundância de servidores do tipo mestre/escravo).

O diferencial da modelagem realizada por Sherlock [Bahl et al. 2007] em relação a outras abordagens, como Shrink [Kandula et al. 2005] e SCORE [Kompella et al. 2005], foi a adição do estado *problemático* em cada nó. Cada nó do GI possui um estado que é representado pela probabilidade do nó estar *ativo*, *problemático* e *inativo*. Dessa forma, um nó com estado $(0,3; 0,5; 0,2)$ indica que o nó possui 30% de chance de estar *ativo*, 50% de chance de estar *problemático* e 20% de chance de estar *inativo*. A soma das probabilidades $P_{\text{ativo}} + P_{\text{problemático}} + P_{\text{inativo}}$ deve ser 1,0.

Assim como no GDS, as arestas no GI são direcionadas e possuem pesos (probabilidades) que representam a força da dependência entre dois nós. Para cada tipo de meta nó, há uma tabela-verdade que indica como as probabilidades de um nó são calculadas a partir das probabilidades de seus antecessores e dos pesos das arestas. As tabelas-verdade para cada tipo de nó estão definidas em [Bahl et al. 2007].

Finalmente, o GI possui dois nós especiais utilizados para possíveis erros no modelo, o nó *sempre problemático* (AT - *always troubled*) e o nó *sempre inativo* (AD - *always down*), que possuem seus estados definidos como $(0; 1; 0)$ e $(0; 0; 1)$, respectivamente. Esses nós são vinculados a cada nó de observação para representar possíveis informações que não estão mapeadas no modelo.

A Figura 3 mostra uma parte de um GI que foi gerado automaticamente por Nemo. Esse grafo foi construído com informações coletadas por 20 agentes instalados na rede da UFMS. O grafo mostra, entre outras coisas, as dependências dos serviços NetBios (137) e FTP (21) do serviço DNS (53). Nesse caso, os nós superiores são os nós raízes de problema, os do meio são os meta nós e os inferiores são os nós de observação.

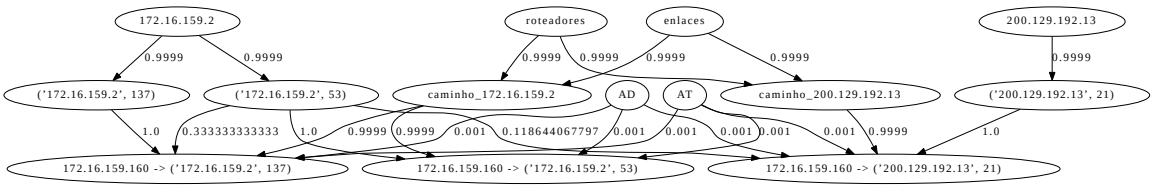

Figura 3. Parte de um GI de uma rede em produção gerado automaticamente por Nemo.

2.3. Redes Bayesianas

Redes Bayesianas são comumente utilizadas para realizar inferências probabilísticas em áreas como Estatística, Inteligência Artificial, Mineração de Dados, etc. Uma Rede Bayesiana (RB) nada mais é que um modelo gráfico que codifica relações probabilísticas entre variáveis de interesse. Sua utilidade advém de sua capacidade de codificar dependências entre todas as variáveis de interesse, de lidar com situações em que nem todas as informações são conhecidas, e de ser útil para se inferir relacionamentos causais. Segue, abaixo, a definição formal de uma Rede Bayesiana.

Definição 1 *Uma Rede Bayesiana $\mathfrak{B}$ é definida como um par $\mathfrak{B} = (G, P)$, em que $G = (V(G), A(G))$ é um grafo direcionado acíclico com conjunto de vértices $V(G) = \{X_1, \ldots, X_n\}$ e conjunto de arestas $A(G) \subseteq V(G) \times V(G)$, e P é uma distribuição de probabilidade conjunta definida nas variáveis correspondentes aos vértices do conjunto $V(G)$ como segue:*

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \pi(X_i))$$

em que $\pi(X_i)$ representa o conjunto de pais (ancestrais diretos) de X_i.

Uma grande contribuição de Sherlock foi a modelagem do problema de diagnóstico de anomalias como um GI. O GI pode ser visto como uma RB, pois é um grafo direcionado acíclico em que os nós representam variáveis aleatórias cujos valores são determinados pelos seus ancestrais diretos de acordo com uma tabela-verdade. Sendo assim, o problema de detecção de anomalias em Sherlock ficou reduzido ao de se explicar inteiramente as variáveis desconhecidas na RB, ou seja, descobrir as probabilidades dos nós raízes de problema que melhor explicam as probabilidades dos nós de observação.

2.3.1. *d*-separação e *d*-conexão

Por tentar explicar todas as variáveis desconhecidas da RB, Sherlock é bastante ineficiente, tanto do ponto de vista de armazenamento quanto do ponto de vista de tempo de execução. Nemo tira vantagem do domínio do problema e tenta explicar somente as variáveis que interferem diretamente em sintomas reportados pelos usuários. Para isso, Nemo explora a propriedade de independência condicional em RBs para descartar variáveis que sejam de fato independentes das variáveis de interesse e que, portanto, não interferem em seus valores. A propriedade de independência condicional em RBs foi bastante estudada no passado, resultando no conceito de *d*-separação. Geiger *et al.* [Geiger et al. 1989] propuseram um algoritmo bastante eficiente, de complexidade linear, para o cálculo da *d*-separação em uma RB.

O conceito de *d*-separação é melhor entendido quando associado ao conceito de *d*-conexão. A definição desses conceitos utiliza os termos nós colisores e nós não colisores. A função desses nós pode ser melhor compreendida pelas representações gráficas mostradas na Figura 4. Nós colisores e não colisores são nós cujos graus de saída são zero e diferente de zero, respectivamente.

Figura 4. Nó v_1 como (a) colisor e (b) não colisor.

Definição 2 *d-**conexão**: Seja G um grafo direcionado em que X, Y e Z são conjuntos disjuntos de vértices de G, então X e Y são d-conectados por Z em G, se, e somente se, existe um caminho não direcionado U entre vértices de X e de Y, tal que para cada nó colisor c em U, c ou um descendente de c está em Z e nenhum nó não colisor pertence a Z.*

Definição 3 *d-**separação**: X e Y são d-separados por Z em G, se, e somente se, eles não são d-conectados por Z.*

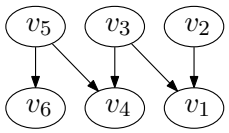

Figura 5. Considere os conjuntos de vértices $X = \{v_1\}$, $Y' = \{v_4\}$, $Y'' = \{v_5, v_6\}$ e $Z = \{v_2, v_3\}$. X e Y' são d-conectados por Z e, X e Y'' são d-separados por Z.

As Definições 2 e 3 formalizam os conceitos de *d*-conexão e *d*-separação, respectivamente, e a Figura 5 mostra um exemplo da utilização desses conceitos em um grafo. Na Seção 2.4, esses conceitos serão utilizados no desenvolvimento de um novo algoritmo que reduz significativamente os GIs gerados por Nemo.

2.4. Redução do Grafo de Inferência

O GI, como proposto por Sherlock, inclui todos os serviços, elementos de hardware e clientes monitorados em uma rede corporativa. Em situações de anomalias, é bem sabido que

apenas parte desses componentes realmente apresenta problemas. Além disso, dependendo da anomalia, poucos clientes são de fato afetados. Como exemplo, pode-se citar um defeito em um roteador que afeta apenas os clientes das sub-redes que são roteadas pelo equipamento defeituoso.

Com base nas observações acima, Nemo procura identificar apenas os nós do GI que influenciam os sintomas observados e descartar os demais nós que são comprovadamente independentes dos sintomas. Para se descartar os nós com a certeza que eles não são os causadores dos sintomas, Nemo utiliza o conceito de *d*-separação.

A questão chave para utilização da *d*-separação para redução do GI é a definição de pelo menos dois dos conjuntos X, Y e Z, pois o terceiro é obtido de modo eficiente (em tempo linear) pelo algoritmo de *d*-separação. Nemo constrói o conjunto Z como sendo os nós de observação que estão reportando problemas, ou seja, os nós sintomas. O conjunto X é definido como sendo os nós raízes de problema que são antecessores dos nós de Z, pois esses são nós que podem afetar os nós sintomas. Os nós de X podem ser obtidos utilizando um algoritmo de busca em largura a partir dos nós sintomas invertendo-se as orientações das arestas. O Algoritmo 1 sumariza essa ideia. Observe que o conjunto Y, obtido pela *d*-separação (Linha 4), contém os nós do GI que são independentes condicionalmente dos sintomas. Para se obter um grafo reduzido R, os nós de Y e suas arestas são removidas do grafo original. R conterá os nós de X, os nós de Z, os nós de observação que são dependentes condicionalmente de Z que podem explicar os sintomas observados, e os meta nós que unem os demais nós do grafo.

Algoritmo 1: Redução (G)

Entrada: Grafo de Inferência G.
Saída: Subgrafo R de G com apenas os nós e arestas que são relevantes para explicação dos sintomas.

```
1 início
2   Z ← Conjunto de nós de observação de G que estão observando degradações
    nos acessos (sintomas);
3   X ← Conjunto de nós raízes de problema de G que são ancestrais dos nós de Z;
4   Y ← d-separação(G, X, Z);  /* Y contém os nós de G tal que,
    X e Y são d-separados por Z em G.  */
5   R ← G − Y;
6   retorna R;
7 fim
```

2.5. Função de Pontuação

O processo de detecção e diagnóstico de anomalias a partir de um GI consiste em encontrar um vetor de atribuição de probabilidades aos estados *ativo, problemático* e *inativo* dos nós raízes de problema que melhor explique os estados observados pelos nós de observação. A ideia básica é propagar os estados (probabilidades) dos nós raízes de problema até os nós de observação e comparar os valores propagados com os estados dos nós de observação obtidos por meio das evidências. As evidências nada mais são que os tempos de acesso aos serviços. Para se determinar o melhor vetor, o algoritmo de inferência utiliza uma função de pontuação que leva em consideração os tempos de resposta dos serviços monitorados pelos agentes.

Em [Bahl et al. 2007], foi observado que os tempos de resposta dos serviços monitorados seguem uma distribuição bimodal em que uma moda caracteriza o tempo médio quando o serviço está normal (*ativo*) e uma segunda moda que caracteriza o tempo médio quando o serviço está enfrentando problemas (*problemático*). Nemo monitorou alguns serviços da rede da UFMS e constatou que a hipótese de distribuição bimodal é de fato verdadeira. A Figura 6 mostra a distribuição de tempos de um dos serviços monitorados. As amostras próximas às modas, quando consideradas separadamente, formam distribuições normais com médias bem próximas às modas. Nemo isola as duas distribuições utilizando o método de clusterização "*k-means*" para encontrar as duas modas e utiliza o ponto médio entre as duas modas para separar as distribuições (ativa e problemática).

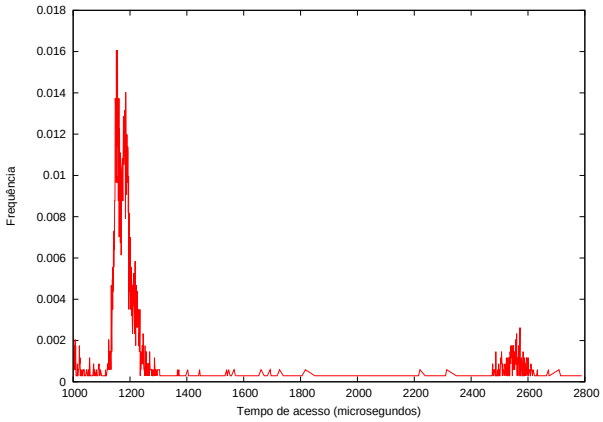

Figura 6. Distribuição de tempos de resposta de um serviço monitorado por Nemo. A primeira moda indica que o serviço está *ativo* e a segunda *problemático*.

Nemo e Sherlock diferem em suas funções de pontuação, pois Sherlock assume e utiliza diretamente a Função de Distribuição de Probabilidade Normal para pontuar um tempo de acesso observado. Isso faz com que tempos de acesso menores que a média recebam pontuações inferiores ao tempo médio ou até mesmo do que tempos ligeiramente superiores à média. Nemo, por outro lado, explora novamente conhecimento específico do domínio do problema para propor uma nova função de pontuação. A função de pontuação de Nemo é discreta e explora a ideia de que tempos de acesso inferiores a média ou ligeiramente superiores devem ter pontuações iguais, pois um tempo de acesso inferior a média indica que o serviço está operando normalmente. Nemo considera apenas os tempos de acesso válidos, tempos muito pequenos, como os decorrentes de *TCP reset*, são descartados. A função de pontuação de Nemo é mostrada no Algoritmo 2. Os resultados de simulação apresentados na Seção 4 mostram que essa função de pontuação aumenta a precisão do algoritmo de inferência nos casos avaliados.

3. Aspectos de Implementação

A ferramenta Nemo, composta de agentes e um gerente, foi implementada e testada na rede de produção da UFMS. Os agentes de Nemo foram desenvolvidos utilizando a linguagem multiplataforma Python com módulos funcionais em ambientes Windows e Linux (netifaces, socket, pcapy, etc.). Além disso, em cada sistema operacional, foi necessário instalar uma biblioteca de captura de pacotes de rede, em Windows a WinPcap e em Linux a LibPcap.

Os agentes de Nemo foram instalados em 20 estações de trabalho Windows. A implantação desses agentes foi feita utilizando a tecnologia de Políticas de Grupo (*Group*

Algoritmo 2: Pontuação (n)

Entrada: Nó de observação n.
Saída: Pontuação em n.

início
  P_{ativo}, $P_{\text{problemático}}$, $P_{\text{inativo}} \leftarrow$ Estado de n;
  μ_{ativo}, $\sigma_{\text{ativo}} \leftarrow$ Média e desvio padrão da distribuição *A*tivo de n;
  se *evidência*$(n) < \mu_{ativo} + \sigma_{ativo}$ **então**
    retorna $P_{\text{ativo}} * 1,0$;
  se *evidência*$(n) < \mu_{ativo} + 2 * \sigma_{ativo}$ **então**
    retorna $P_{\text{ativo}} * 0,7 + P_{\text{problemático}} * 0,25 + P_{\text{inativo}} * 0,05$;
  se *evidência*$(n) < \mu_{ativo} + 3 * \sigma_{ativo}$ **então**
    retorna $P_{\text{ativo}} * 0,4 + P_{\text{problemático}} * 0,5 + P_{\text{inativo}} * 0,1$;
  se *evidência*$(n) < \mu_{ativo} + 4 * \sigma_{ativo}$ **então**
    retorna $P_{\text{ativo}} * 0,05 + P_{\text{problemático}} * 0,75 + P_{\text{inativo}} * 0,2$;
  retorna $P_{\text{problemático}} * 0,8 + P_{\text{inativo}} * 0,2$;
fim

Policy) do AD (*Active Directory*) em execução no servidor de domínio com Windows Server 2008. Para a realização da instalação distribuída via AD, foi criado um pacote executável MSI (*Microsoft Software Installer*), exigência do ambiente, utilizando o *framework* py2exe do Python.

O gerente foi desenvolvido em Python e C++, de tal forma que a parte de comunicação e a interface foram desenvolvidas em Python e a parte de processamento em C++. Essa separação foi necessária devido ao enorme *overhead* existente em Python e que tornava o processamento extremamente lento. Para a construção do GI foi utilizada a biblioteca de estruturas de dados Lemon que é compatível com ambas as linguagens. Além disso, as trocas de mensagens entre os agentes e o gerente utilizam um protocolo simples de aplicação, criado para enviar dados de atualização do GDS e requisições de execução do processo de diagnóstico de anomalias. Os grafos de inferência, como o da Figura 3, foram gerados utilizando a biblioteca Graphviz.

Um ponto interessante da ferramenta, e que foi observado durante os testes do sistema, é que ela detectou um serviço desconhecido pela equipe de administração da rede. Um usuário havia instalado um Gerente SNMP em um Câmpus da instituição e não havia comunicado à equipe de suporte. Muito embora esse não seja o objetivo da ferramenta, ela pode ser utilizada para detectar serviços que ferem a política de uso da rede como, por exemplo, comunidades P2P e serviços não homologados.

4. Avaliação Experimental

Nesta seção, são apresentados resultados de simulação para avaliar vários aspectos de desempenho de Nemo. Muito embora Nemo tenha sido implementado e testado em um ambiente de produção, a avaliação por simulação permite que cenários maiores e mais complexos sejam explorados, além de se tratar de um ambiente controlado e passível de reprodução dos resultados. Inicialmente, é avaliado o algoritmo de redução do grafo de inferência. Este resultado é importante, pois o algoritmo de redução pode ser utilizado por outras abordagens que utilizam grafos de inferência para detecção de anomalias. Em seguida, Nemo é comparado com Sherlock e Spotlight. Spotlight [John et al. 2010] é uma proposta

mais recente e bem mais rápida que Sherlock. Entretanto, Spotlight utiliza uma heurística para reduzir o GI em um grafo bipartido que não preserva as propriedades teóricas do modelo inicial. Além disso, os resultados de precisão de Spotlight são bem inferiores aos de Sherlock. As métricas utilizadas para comparação são: precisão e tempo de execução. As próximas seções descrevem a metodologia de avaliação e apresentam os resultados numéricos.

4.1. Metodologia de Avaliação

Para a avaliação, foram gerados aleatoriamente diversos grafos de inferência correspondentes a topologias de rede, dependências de serviços e distribuições de tempo de acesso. A metodologia utilizada para geração do grafo é a mesma utilizada em [Bahl et al. 2007, John et al. 2010]. Os tempos de acesso são gerados de forma aleatória utilizando distribuições Normais com médias (ativo e problemático) e desvios padrão obtidos a partir de dados reais coletados pelos agentes de Nemo na rede da UFMS. As probabilidades de dependência são geradas aleatoriamente com distribuição uniforme.

As simulações foram realizadas em uma única máquina com dois processadores Intel Xeon E5530, 32GB de RAM e sistema operacional Linux com kernel 3.2.0. Todos os resultados foram obtidos a partir da média de pelo menos 50 execuções independentes. Além disso, um intervalo de confiança de 95% foi calculado para cada um dos resultados.

Para se avaliar a precisão, anomalias devem ser injetadas artificialmente nos grafos de inferência. Uma anomalia é introduzida selecionando-se aleatoriamente um nó raiz de problema e marcando-o como problemático. Após essa escolha, a anomalia deve ser propagada e refletida nos nós de observação. A propagação é feita por meio de um passeio probabilístico no grafo de inferência utilizando-se as probabilidades de dependência entre os nós. Observe que uma anomalia injetada em um nó raiz de problema pode refletir em vários nós de observação. Um algoritmo de inferência acerta quando indica corretamente os nós raízes de problema que foram marcados inicialmente como problemáticos.

Os grafos de inferência foram gerados com quantidades de nós raízes de problema que variaram de 100 a 20.000 para se avaliar o algoritmo de redução e de 150 a 250 para se avaliar a precisão e o tempo de execução. A quantidade de anomalias injetadas variaram de 1 a 4.

4.2. Redução do Grafo de Inferência

O Algoritmo 1, apresentado na Seção 2.4, é executado para reduzir a quantidade de nós do GI com o objetivo de se reduzir o tempo de execução do processo de diagnóstico de anomalias. Essa redução faz com que somente nós que estão relacionados às anomalias permaneçam no grafo e, consequentemente, que os nós que não auxiliam na detecção sejam descartados.

Os grafos utilizados na avaliação do algoritmo foram gerados variando-se as quantidades de anomalias injetadas e de nós raízes de problema. Para cada combinação, foram gerados 100 grafos diferentes utilizando-se a metodologia descrita na Seção 4.1. Pode-se observar na Figura 7 que a taxa de redução apresentada é diretamente proporcional a quantidade de nós raízes de problema e inversamente proporcional a quantidade de anomalias. A taxa de redução variou entre 50% até pouco mais de 99%. Essa redução elevada é explicada pela alta quantidade de nós raízes de problema que não explicam os sintomas provocados pelas anomalias. A figura também mostra os intervalos de confiança de 95% para a média.

Muito embora a redução seja significativa, ela será útil somente se o processo de diagnóstico não for afetado. A seção seguinte mostra que o algoritmo de redução não interfere

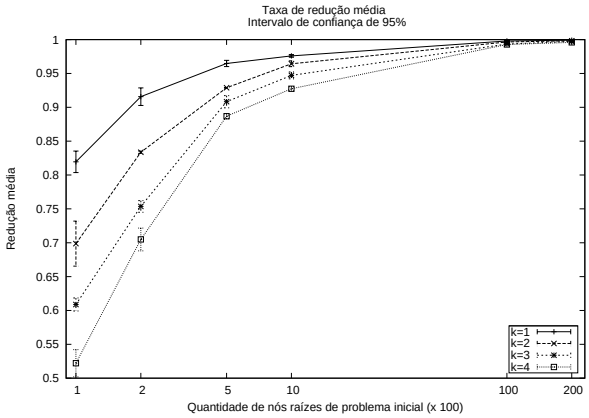

Figura 7. Taxa de redução média de nós raízes de problema, com um intervalo de confiança de 95%, de grafos com diferentes quantidades (100**,** 200**,** 500**,** $1k$**,** $10k$ **e** $20k$**) de nós raízes de problema e com** 1 **até** 4 **anomalias simultâneas.**

na precisão do processo de detecção de anomalias.

4.3. Precisão

A precisão de uma abordagem é definida como a razão entre a quantidade de execuções com as detecções corretas pela quantidade total de execuções. A Figura 8 mostra a precisão das abordagens com GIs de quantidade fixa de nós de observação, diferentes quantidades de nós raízes de problema e anomalias simultâneas. Nas comparações, Nemo se mostrou superior a Sherlock [Bahl et al. 2007] e ao Spotlight [John et al. 2010] em todos os cenários estudados.

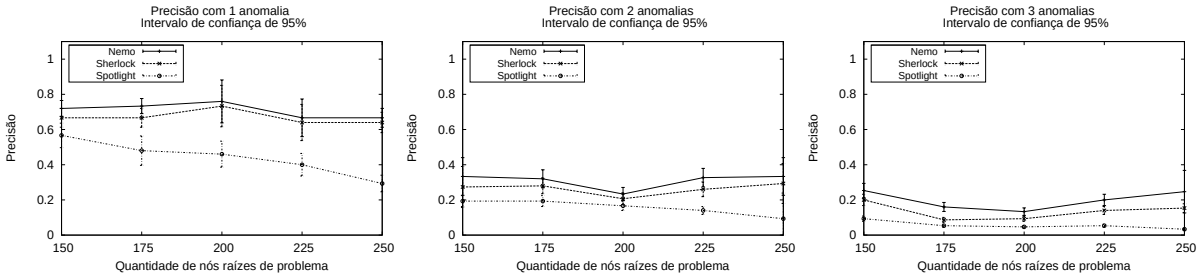

Figura 8. Comparação da precisão das abordagens com uma, duas e três anomalias simultâneas, variando a quantidade de nós raízes de problema.

Muito embora os valores de precisão para duas e três anomalias sejam baixos, os resultados apresentados consideram como acerto somente quando todas as anomalias estão no vetor de atribuição selecionado como melhor pelo algoritmo de inferência. Entretanto, Nemo identifica corretamente pelo menos uma das anomalias em mais de 82% dos casos, conforme mostra a Tabela 1. Quando se considera os cinco melhores vetores, Nemo acerta pelo menos uma das anomalias em mais de 97% dos casos.

A verificação de hipóteses com cinco diferentes pontuações é totalmente plausível por administradores de rede em um processo de diagnóstico de anomalias em redes corporativas. O processo pode ser desenvolvido de forma iterativa, corrigindo-se inicialmente as anomalias que foram indicadas corretamente e repetindo-se o processo até que todas sejam resolvidas.

Tabela 1. Percentual de detecção correta de pelo menos uma anomalia.

Quantidade de nós raízes de problema	Percentual de detecção correta					
	2 anomalias			3 anomalias		
	Nemo	Sher.	Spot.	Nemo	Sher.	Spot.
150	82,0	79,3	73,3	97,3	95,3	91,3
175	87,3	84,6	64,6	94,6	91,3	72,6
200	91,3	87,3	65,3	95,3	92,0	78,6
225	92,0	90,6	62,0	92,6	91,3	67,3
250	91,3	90,0	58,0	94,6	93,3	60,0

4.4. Tempo de execução

A Tabela 2 compara os tempos de execução das três abordagens quando executadas com os grafos de inferência utilizados na Seção 4.2. Como esperado, Nemo é bem mais rápido que Sherlock em todos os casos. Entretanto, Spotlight é mais rápido que Nemo quando duas e três anomalias são injetadas. Os baixos tempos de execução de Spotlight decorrem de uma abordagem bem diferente das demais e que é incapaz de produzir múltiplas hipóteses, além de ser bem menos precisa. O algoritmo de redução também foi testado com Spotlight, mas não resultou em redução significativa dos tempos de execução.

Tabela 2. Tempo médio de execução das três abordagens.

Quantidade de nós raízes de problema	Tempo médio de execução (s)								
	1 anomalia			2 anomalias			3 anomalias		
	Nemo	Sher.	Spot.	Nemo	Sher.	Spot.	Nemo	Sher.	Spot.
150	0,0028	0,0118	0,0064	0,0858	2,5260	0,0026	6,2356	653,4461	0,0228
175	0,0018	0,0180	0,0016	0,1162	3,9704	0,0034	6,7199	1548,27	0,0560
200	0,0032	0,0196	0,0004	0,1188	4,7354	0,0040	7,4836	3853,56	0,0584
225	0,0038	0,0198	0,0054	0,1104	6,1285	0,0046	24,1810	8289,36	0,0570
250	0,0056	0,0236	0,0050	0,1210	7,8026	0,0062	32,4695	14960,9	0,0672

5. Conclusão

Neste artigo é proposta e avaliada uma ferramenta para detecção e diagnóstico de anomalias. A ferramenta é não intrusiva, portanto não modifica aplicações, servidores ou sistemas legados. Ela utiliza somente traços de rede para inferir as possíveis causas dos problemas, sendo particularmente útil para ambientes legados. O código fonte da ferramenta está disponível em http://ndsg.facom.ufms.br/nemo. Uma funcionalidade ainda não implementada e testada por completo é uma interface gráfica para os usuários poderem alterar parâmetros e visualizar os resultados em tempo real.

As principais contribuições deste trabalho são: um algoritmo eficiente para redução de grafos de inferência e uma nova função de pontuação para inferência de anomalias. O algoritmo se mostrou bastante eficiente para redução não somente do grafo como também do tempo de execução da ferramenta e a função de pontuação melhorou a precisão do diagnóstico em todos os cenários avaliados.

Referências

Bahl, P., Chandra, R., Greenberg, A. G., Kandula, S., Maltz, D. A., and Zhang, M. (2007). Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of The 2007 ACM SIGCOMM Conference on Applications,*

Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'07), pages 13–24, New York, NY, USA.

Barham, P., Black, R., Goldszmidt, M., Isaacs, R., MacCormick, J., Mortier, R., and Simma, A. (2008). Constellation: Automated Discovery of Service and Host Dependencies in Networked Systems. *Technical Report, MSR-TR-2008-67, Microsoft Research*.

Barham, P., Donnelly, A., Isaacs, R., and Mortier, R. (2004). Using Magpie for Request Extraction and Workload Modelling. In *Proceedings of The 6th Symposium on Operating Systems Design and Implementation (OSDI'04)*, pages 259–272, Berkeley, CA, USA.

Chen, M. Y., Accardi, A., Kiciman, E., Patterson, D. A., Fox, A., and Brewer, E. A. (2004). Path-Based Failure and Evolution Management. In *Proceedings of The 1st Symposium on Networked Systems Design and Implementation (NSDI'04)*, pages 309–322, Berkeley, CA, USA.

Fonseca, R., Porter, G., Katz, R. H., Shenker, S., and Stoica, I. (2007). X-Trace: A Pervasive Network Tracing Framework. In *Proceedings of The 4th Symposium on Networked Systems Design and Implementation (NSDI'07)*, Berkeley, CA, USA.

Galstad, E. (1999). Nagios. `http://www.nagios.org`.

Geiger, D., Verma, T., and Pearl, J. (1989). d-Separation: From Theorems to Algorithms. In *UAI*, pages 139–148.

HP (1990). HP Openview. `http://www.openview.hp.com`.

IBM (1996). IBM Tivoli. `http://www.ibm.com/tivoli`.

John, D., Prakash, P., Kompella, R. R., and Chandra, R. (2010). Shedding Light on Enterprise Network Failures Using Spotlight. In *Proceedings of The 29th IEEE Symposium on Reliable Distributed Systems (SRDS'10)*, pages 167–176.

Kandula, S., Katabi, D., and Vasseur, J. P. (2005). Shrink: A Tool For Failure Diagnosis in IP Networks. In *Proceedings of MineNet Workshop at 2006 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'06)*, pages 173–178.

Kompella, R. R., Yates, J., Greenberg, A. G., and Snoeren, A. C. (2005). IP Fault Localization Via Risk Modeling. In *Proceedings of The 2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*.

Lyon, G. (1997). Nmap Security Scanner. `http://nmap.org`.

Natarajan, A., Ning, P., Liu, Y., Jajodia, S., and Hutchinson, S. E. (2012). NSDMiner: Automated discovery of Network Service Dependencies. In *Proceedings of The IEEE (INFOCOM'12)*, pages 2507–2515.

Reynolds, P., K., C. E., Wiener, J. L., Mogul, J. C., Shah, M. A., and Vahdat, A. (2006). Pip: Detecting the Unexpected in Distributed Systems. In *Proceedings of The 3rd Symposium on Networked Systems Design and Implementation (NSDI'06)*, Berkeley, CA, USA.

Tak, B. C., Tang, C., Zhang, C., Govindan, S., Urgaonkar, B., and Chang., R. N. (2009). vPath: Precise Discovery of Request Processing Paths from Black-Box Observations of Thread and Network Activities. In *Proceedings of The 34th USENIX Annual Technical Conference (USENIX'09)*, Berkeley, CA, USA.