UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

OTTO JULIO AHLERT PINNO DA SILVA

Detecção de anomalias em aplicações Web utilizando filtros baseados em coeficiente de correlação parcial

Goiânia
2014

OTTO JULIO AHLERT PINNO DA SILVA

Detecção de anomalias em aplicações Web utilizando filtros baseados em coeficiente de correlação parcial

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Kleber Vieira Cardoso

Co-Orientadora: Profa. Dra. Sand Luz Corrêa

Goiânia
2014

OTTO JULIO AHLERT PINNO DA SILVA

Detecção de anomalias em aplicações Web utilizando filtros baseados em coeficiente de correlação parcial

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Computação, aprovada em 31 de Outubro de 2014, pela Banca Examinadora constituída pelos professores:

Prof. Dr. Kleber Vieira Cardoso
Instituto de Informática – UFG
Presidente da Banca

Profa. Dra. Sand Luz Corrêa
Instituto de Informática – UFG

Prof. Dr. Vagner José do Sacramento Rodrigues
Instituto de Informática – UFG

Prof. Dr. Aldri Luiz dos Santos
Instituto de Informática – UFPR

Otto Julio Ahlert Pinno da Silva

Graduado em Desenvolvimento de Software pela Faculdade de Ciências Jurídicas e Sociais Aplicadas do Araguaia (2007-2009). Graduado em Ciência da Computação pela Universidade Federal de Mato Grosso (2008-2011). Pós-graduado em Redes de Computadores pelo Instituto MT de Pós-graduação (2010-2010). No período do mestrado, atuou como bolsista do Projeto FIBRE desenvolvido no INF/UFG em parceria com outras instituições nacionais e estrangeiras.

Resumo

da Silva, Otto Julio Ahlert Pinno. **Detecção de anomalias em aplicações Web utilizando filtros baseados em coeficiente de correlação parcial**. Goiânia, 2014. 80p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Encontrar falhas ou causas de problemas de desempenho em sistemas computacionais *Web* atuais é uma tarefa árdua que envolve muitas horas de análise de *logs* e métricas de sistemas. Para ajudar administradores nessa tarefa, diversos mecanismos de detecção de anomalia foram propostos visando analisar o comportamento do sistema mediante a coleta de um grande volume de informações estatísticas que demonstram o estado e o desempenho do sistema computacional. Uma das abordagens adotadas por esses mecanismo é o monitoramento por meio de correlações fortes identificadas no sistema. Nessa abordagem, a coleta desse grande número de dados gera inconvenientes associados à comunicação, armazenamento e, especialmente, com o processamento das informações coletadas. Apesar disso, poucos mecanismos de detecção de anomalias possuem uma estratégia para a seleção das informações estatísticas a serem coletadas, ou seja, para a seleção das métricas monitoradas. Este trabalho apresenta três filtros de seleção de métricas para mecanismos de detecção de anomalias baseados no monitoramento de correlações. Esses filtros foram baseados no conceito de correlação parcial, técnica que é capaz de fornecer informações não observáveis por métodos de correlações comuns. A validação desses filtros foi realizada sobre um cenário de aplicação *Web*, sendo que, para simular esse ambiente, nós utilizamos o TPC-W, um *Benchmark* de transações *Web* do tipo *E-commerce*. Os resultados obtidos em nossa avaliação mostram que um de nossos filtros permitiu a construção de uma rede de monitoramento com 8% menos métricas que filtros estado-da-arte, além de alcançar uma cobertura de falhas até 10% mais eficiente.

Palavras–chave

Detecção de anomalias, Sistemas distribuídos complexos, Correlação parcial, Filtragem de métricas.

Abstract

da Silva, Otto Julio Ahlert Pinno. **Anomaly Detection in Web applications using filters based on partial correlation coefficient**. Goiânia, 2014. 80p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Finding faults or causes of performance problems in modern Web computer systems is an arduous task that involves many hours of system metrics monitoring and log analysis. In order to aid administrators in this task, many anomaly detection mechanisms have been proposed to analyze the behavior of the system by collecting a large volume of statistical information showing the condition and performance of the computer system. One of the approaches adopted by these mechanism is the monitoring through strong correlations found in the system. In this approach, the collection of large amounts of data generate drawbacks associated with communication, storage and specially with the processing of information collected. Nevertheless, few mechanisms for detecting anomalies have a strategy for the selection of statistical information to be collected, i.e., for the selection of monitored metrics. This paper presents three metrics selection filters for mechanisms of anomaly detection based on monitoring of correlations. These filters were based on the concept of partial correlation technique which is capable of providing information not observable by common correlations methods. The validation of these filters was performed on a scenario of Web application, and, to simulate this environment, we use the TPC-W, a Web transactions Benchmark of type E-commerce. The results from our evaluation shows that one of our filters allowed the construction of a monitoring network with 8% fewer metrics that state-of-the-art filters, and achieve fault coverage up to 10% more efficient.

Keywords

Anomaly detection, Complex distributed systems, Partial correlation, Metric filtering.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

EB *Emulated Browser*.

MST *Minimum Spanning Tree*.
MVA *Mean Value Analysis*.

NTP *Network Time Protocol*.

PCTN *Partial Correlation Threshold Network*.
PRDP Porcentagem de Resíduos Discrepantes Permitidos.

TCP *Transmission Control Protocol*.

CAPÍTULO 1

Introdução

Tendências como computação em nuvem e serviços *online* têm ampliado a demanda por infraestruturas computacionais de grande porte. Essa demanda tem sido uma das responsáveis pelo surgimento de centros de dados (*data centers*), onde recursos computacionais como servidores, equipamentos de rede e sistemas de armazenamento de dados são fornecidos através de recursos virtuais e acessos remotos [41].

Centros de dados podem ser construídos objetivando tanto o uso de uma empresa, como também o fornecimento de recursos a terceiros por meio de contratos. O último caso permite que interessados obtenham acesso a recursos computacionais sem que haja a necessidade de assumir elevados custos com aquisição e manutenção de infraestruturas de computação e refrigeração. Contudo, independentemente do tipo de serviço fornecido ou a quem se destina, os centros de dados são conhecidos por possuírem alta complexidade [47]. Esse fato pode ser explicado, em parte, pelo grande número de equipamentos que normalmente constituem esses ambientes e a grande quantidade e variedade de aplicações que costumam ser executadas nos servidores.

Tipicamente, centros de dados executam aplicações *Web* como, por exemplo, serviços de comércio eletrônico, redes sociais e serviços de transmissão de conteúdo multimídia [47]. Sistemas de software dessa natureza chegam a prover serviços para centenas de milhões de usuários na Internet [32, 33, 34]. Outra característica comum a esses sistemas é a conformidade com uma arquitetura orientada a serviços. Nesse tipo de arquitetura, diversos componentes de software são distribuídos em várias camadas e combinados para compor as funcionalidades de uma aplicação maior. Cada componente de software é uma unidade autocontida, com especificidades de implementação, tecnologia e configuração independentes dos demais. Além disso, esses componentes podem estar distribuídos em diferentes servidores ou equipamentos, alojados em um ou mais centros de dados. Finalmente, fatores observados apenas em tempo de execução, como sequência de acesso, concorrência e exaustão de recursos, podem levar a aplicação a se comportar de maneira não prevista em tempo de projeto [44]. Portanto, em aplicações *Web*, a interação entre os próprios componentes da aplicação é complexa, como também é complexa a interação entre esses componentes e o ambiente de execução. Por essa razão, aplicações *Web* são

frequentemente caracterizadas como sistemas distribuídos complexos [10, 23, 51, 36, 48].

Embora a complexidade de grandes aplicações *Web* seja bem conhecida, é esperado que esse tipo de aplicação apresente alta disponibilidade, confiabilidade e responsividade, sob pena de causar perdas expressivas tanto para os usuários da infraestrutura quanto para seus provedores. Para ilustrar algumas perdas relevantes causadas por falhas em ambientes de centros de dados, em agosto de 2009, o PayPal, um sistema de pagamentos *online* bastante popular, ficou inacessível durante uma hora e gerou prejuízo de 7,2 milhões de dólares ao comércio eletrônico [42]. Em junho de 2010, a Amazon.com registrou três horas de desempenho instável no seu serviço de comércio eletrônico. Páginas de produtos levavam minutos para serem carregadas e buscas por itens retornavam vazias. Vários clientes não conseguiam concluir um pedido ou compra. Estima-se que esse incidente resultou em um custo de 1,75 milhões de dólares por hora para a empresa [31, 47]. Em fevereiro de 2012, uma data incorreta impediu a criação de um certificado durante a inicialização de um componente da plataforma Microsoft Azure. Essa falha desencadeou uma série de eventos que levaram a plataforma de computação em nuvem da Microsoft a ficar inoperante durante 24 horas. Aplicações de várias empresas usuárias da plataforma ficaram indisponíveis durante esse período [26].

Nesse contexto, é nítido e relevante que a interrupção de aplicações *Web* tem um custo elevado. Portanto, é importante identificar quais partes do processo de detecção e correção de falhas são mais críticas e investir em seu aperfeiçoamento. Estudos revelam que o esforço gasto para detectar as causas de uma falha em um sistema computacional corresponde a 75% do esforço total despendido para recuperá-lo e retorná-lo à operação normal [9]. Estudos também mostram que falhas são frequentemente precedidas ou seguidas por anomalias [5], ou seja, situações que desviam da definição de comportamento normal. Logo, detectar anomalias em sistemas distribuídos grandes e complexos, como aplicações *Web* hospedadas em centros de dados, pode acelerar a descoberta de falhas ou mesmo antecipá-las, melhorando a confiabilidade dessas aplicações.

Nos últimos anos, diversas ferramentas e abordagens de detecção de anomalias em sistemas distribuídos complexos foram propostas tanto na indústria como na academia. Algumas abordagens consistem em ferramentas comerciais de monitoramento, como *SmartCloud* da IBM [21] e o *System Center Operations Manager* da Microsoft [30]. Nessas ferramentas, os administradores configuram manualmente regras que acionam a coleta de dados e ativam mecanismos de alerta sempre que o limiar, definido na regra, for atingido. Outras abordagens utilizam modelos analíticos específicos para capturar correlações entre métricas monitoradas. Quando a relação descrita nos modelos é violada, uma anomalia é detectada. Os modelos analíticos mais comuns incluem regressão [22, 23, 36] e mistura de Gaussianas [20]. Existem também abordagens que analisam o comportamento das aplicações através do rastreamento da interação dos seus componentes [3, 6, 9],

enquanto outros trabalhos capturam a assinatura de falhas conhecidas [7, 19]. Por fim, existem abordagens que fazem uso de aprendizado de máquina para capturar o inter-relacionamento entre os componentes das aplicações e o ambiente em que elas executam [12, 45, 48, 54, 55].

1.1 Motivação

Uma abordagem comumente utilizada por vários mecanismos de detecção de anomalias é o monitoramento baseado na correlação entre métricas. No contexto desses mecanismos, o termo métrica refere-se a um conjunto de mensurações relacionadas com o estado e o desempenho de um sistema computacional. Essas mensurações são coletadas periodicamente para aferir a "saúde" do sistema. Uma questão essencial em muitos mecanismos de detecção de anomalias é a seleção das métricas que serão monitoradas. Alguns mecanismos assumem um conjunto fixo e pequeno de métricas, enquanto outros não se preocupam em realizar uma seleção prévia das métricas monitoradas. Ambas as estratégias, no entanto, apresentam desvantagens. Monitorar um número pequeno de métricas dificulta não apenas a detecção das anomalias, como também uma análise mais detalhada de suas causas. Por outro lado, não selecionar previamente um conjunto de métricas, pode levar a um número excessivo de mensurações que consomem diversos recursos da infraestrutura, como memória, disco e rede. Além disso, o tratamento regular de um volume desnecessariamente alto de dados demanda tempo e recursos sem, necessariamente, melhorar a acurácia. Como ilustração, um pequeno sistema de comércio eletrônico abrange tipicamente componentes como um servidor *Web*, um servidor de aplicação e um servidor de banco de dados. Cada um desses componentes, por sua vez, pode conter centenas de métricas. Adicionalmente, é relevante coletar métricas também da plataforma na qual o sistema é executado, ou seja, do sistema operacional e do hardware. Naturalmente, o problema escala em ordens de grandeza quando se trata de um centro de dados com uma enorme quantidade de recursos virtuais e físicos. Apesar de sua importância, a seleção de métricas tem sido negligenciada por muitos trabalhos que abordam detecção de anomalias [36].

1.2 Objetivos

Neste trabalho, apresentamos três filtros de métricas monitoradas por mecanismos de detecção de anomalias que atuam sobre ambientes de aplicações *Web*. O objetivo é selecionar automaticamente um conjunto reduzido de métricas que, ao serem rastreadas, forneçam uma cobertura adequada de falhas sem incorrer em um custo elevado de monitoramento. Para selecionar o conjunto reduzido de métricas, este trabalho usa o conceito

de *correlação parcial* [2]. A correlação parcial descreve o quanto o relacionamento entre duas variáveis é resultado de suas correlações com uma variável intermediária. Diferentemente de técnicas tradicionais que analisam o relacionamento entre pares de variáveis, ela tem sido empregada em áreas como Economia [25] e Bioinformática [15] para construir uma rede de correlações onde é possível mostrar a influência de algumas variáveis na estrutura de correlação de outras. Isso permite excluir relacionamentos que são resultados de correlações indiretas. Avaliamos nossas propostas em um ambiente real onde implantamos um pequeno serviço de comércio eletrônico e emulamos transações *Web* através do TPC^{TM} *Benchmark* W (TPC-W) [29], desenvolvido pelo *Transaction Processing Performance Council* em 2002. Os resultados obtidos mostram que uma de nossas propostas conseguiu identificar um conjunto reduzido de métricas que são capazes de obter resultados semelhantes aos obtidos pelas abordagens estado-da-arte.

Munawar et al. [36] apresenta algumas metodologias para filtragem de métricas e correlações monitoradas. No entanto, como mencionado anteriormente, a correlação parcial possui características interessantes que, até onde temos conhecimento, não foram avaliadas por outros trabalhos na seleção de métricas e correlações monitoradas em sistemas computacionais *Web*.

Vale ressaltar que, apesar termos avaliado os filtros propostos neste trabalho apenas em um sistema computacional *Web*, em teoria, os filtros baseados em correlação parcial poderiam ser aplicados na seleção de métricas e correlações para o monitoramento de qualquer sistema computacional.

1.3 Contribuições

As principais contribuições deste trabalho são:

- Proposta de três métodos baseados em correlação parcial para filtrar métricas monitoradas em sistemas distribuídos complexos, como sistemas *Web*;
- Implementação e avaliação dos métodos propostos com base em dados coletados de um sistema real de comércio eletrônico;
- Comparação dos métodos propostos com outros filtros de métricas existentes na literatura, como filtro baseado em coeficiente correlação de Pearson e árvore geradora mínima – *Minimum Spanning Tree* (MST).
- Obtenção de uma rede de monitoramento com uma menor quantidade de correlações e métricas que abordagens estado-da-arte, tendendo a reduzir o consumo de processamento no monitoramento.
- Redes com resultados próximos nas avaliações e, em alguns casos, superiores aos obtidos por abordagens estado-da-arte.

1.4 Organização da dissertação

O restante desta dissertação está organizado da seguinte forma:

- Capítulo 2: introduz os fundamentos nos quais este trabalho é baseado, apresentando os conceitos principais relacionados à área de detecção de anomalias em sistemas computacionais complexos por meio do uso de correlações. Além disso, é apresentada uma revisão sobre trabalhos relevantes da literatura que tratam o problema e também trabalhos que se concentram em filtragem de métricas;
- Capítulo 3: descreve como usamos o conceito de correlação parcial para criar um filtro automático de métricas que pode ser utilizado por mecanismos de detecção de anomalias em sistemas computacionais *Web*.
- Capítulo 4: apresenta a infraestrutura utilizada para validar nossa proposta, os sistemas de software implantados nessa infraestrutura e a metodologia de testes usada para geração de carga e injeção de falhas. Além disso, esse capítulo mostra também os resultados obtidos na avaliação experimental;
- Capítulo 5: apresenta as conclusões e discute as perspectivas para trabalhos futuros.

CAPÍTULO 2

Fundamentação e trabalhos relacionados

Neste capítulo, discutimos os conceitos principais da área de detecção de anomalias, correlações de Pearson, correlação parcial e, por fim, apresentamos uma revisão sobre trabalhos relevantes que abordam esse tema no contexto de aplicações *Web*.

2.1 Terminologia

2.1.1 Aplicações Web

Aplicações *Web*, tipicamente, executam sobre arquiteturas multicamadas compostas por servidor *Web*, servidor de aplicação e servidor de banco de dados, como ilustrado na Figura 2.1. Cada camada fornece informações para a camada superior e requisita informações da camada inferior. Além disso, com o objetivo de tornar a arquitetura escalável, cada camada pode ser composta por mais de um servidor, necessitando de balanceadores de carga para distribuir as requisições recebidas entre os diversos servidores. Vale ressaltar que os servidores de uma mesma camada não necessariamente devem ser homogêneos, ou seja, esses servidores podem possuir diferentes características.

2.1.2 Falha

No contexto deste trabalho, usamos o termo falha para denotar um comportamento observável de um sistema computacional que não corresponde à especificação desse sistema [47]. Algumas falhas típicas de sistemas computacionais incluem: laços infinitos, vazamento de memória, erro em operações de E/S e falhas de comunicação ou rede [37]. Adicionalmente, falhas em sistemas computacionais podem ser classificadas em: permanentes, transientes e intermitentes. Falhas permanentes persistem até que o sistema seja reparado. Um exemplo comum é uma desconexão causada por um cabo de rede defeituoso. Falhas transientes eventualmente desaparecem sem nenhuma intervenção aparente, enquanto que falhas intermitentes são falhas transientes que aparecem de

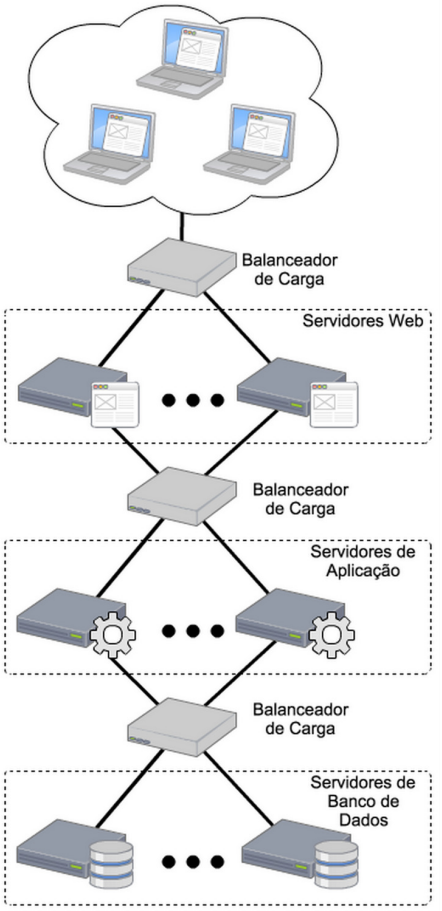


Figura 2.1: *Estrutura multicamada de servidores Web.*

forma recorrente. Um exemplo comum de falha transiente é a causada por sobrecarga do sistema.

De maneira geral, as falhas em sistemas computacionais são causadas por: (1) software, (2) hardware, (3) erro do operador ou administrador ou (4) por violação de segurança [47]. Falhas causadas por software decorrem normalmente de incidentes envolvendo rotinas de manutenção, atualização de software, integração de sistemas e rotinas complexas de recuperação de sistemas. Falhas causadas por hardware envolvem erros em dispositivos físicos e agentes ambientais como calor, umidade, desastres naturais, dentre outros. Erros causados por operadores decorrem normalmente de configurações de parâmetros incorretos, como, por exemplo, a especificação de um número insuficiente de *threads* para atender as requisições de um servidor *Web*. Finalmente, falhas causadas por violação de segurança envolvem incidentes de natureza maliciosa, como ataque de negação de serviço, vírus, exploração de vulnerabilidades de sistemas e aplicações, etc.

No escopo deste trabalho, consideramos falhas causadas apenas por incidentes de natureza não maliciosa, ou seja, consideramos apenas falhas decorrentes de problemas em software, hardware e erro de operação.

2.1.3 Anomalia

Anomalias são perturbações em comportamentos esperados e podem estar presentes nos mais diversos ambientes. Em sistemas computacionais, por exemplo, uma anomalia pode se manifestar através do consumo excessivo de um ou mais recursos computacionais (CPU, memória, disco ou rede), lentidão no processamento de requisições, alterações frequentes de rotas em um roteador, padrões de tráfego de rede diferentes dos usuais, dentre outros eventos. Essas perturbações são frequentemente ocasionadas por falhas [5]. Portanto, a detecção de anomalias é uma atividade essencial para melhorar a confiabilidade dos sistemas computacionais.

Tipicamente, um mecanismo de detecção de anomalias busca encontrar padrões de comportamento que diferem do comportamento esperado para o sistema. Como, em geral, sistemas de software são testados antes de serem implantados, é razoável assumir que esses sistemas proveem serviços corretamente a maior parte das vezes. Portanto, mecanismos de detecção de anomalias monitoram regularmente o sistema e assumem o comportamento mais frequente como o normal. Esse comportamento é, então, usado como base de comparação para detectar comportamentos anômalos. No entanto, existem várias dificuldades que tornam a construção de um mecanismo de detecção de anomalias uma tarefa complexa. As principais dificuldades [5] são descritas a seguir:

- Monitoramento: para determinar se um comportamento é anômalo, é necessário capturar o comportamento corrente e compará-lo com o esperado. Tanto o comportamento corrente como o esperado são capturados através de mecanismos de monitoramento. Uma questão essencial é a decisão a respeito de quais métricas devem ser monitoradas. Essa decisão tem impacto tanto na acurácia e efetividade do mecanismo, como também na sobrecarga que o mecanismo impõe ao restante do sistema.
- Fronteira entre normalidade e anomalia: a diversidade de componentes monitorados e as variações que podem ocorrer entre condições semelhantes do sistema tornam difícil a definição do que pode ser considerado comportamento normal. Muitas vezes, também acontece das fronteiras entre comportamento normal e comportamento anômalo estarem muito próximas, dificultando a diferenciação entre os dois tipos de comportamentos.
- Evolução dos sistemas: com o tempo, é comum os sistemas evoluírem, ganharem mais usuários e terem mais informações em suas bases de dados. Nesses casos, tempos de respostas e padrões de utilização de recursos considerados normais no passado podem passar a ser considerados anormais após a evolução do sistema.
- Base de dados de treinamento: muitos mecanismos de detecção de anomalias necessitam de uma base de dados a partir da qual seja possível extrair o comportamento

mais frequente ou normal. Em muitos casos, construir essa base de dados é um grande desafio. Outra dificuldade em relação a bases de dados de treinamento é a presença de ruídos nos dados. Existe uma tendência de se confundir ruído com anomalia e os mecanismos devem estar preparados para distinguir os dois fenômenos.

Neste trabalho, tratamos especificamente das dificuldades relacionadas à atividade de monitoramento realizada pelos mecanismos de detecção de anomalias.

2.1.4 Detecção

A solução de problemas em sistemas computacionais envolve tipicamente três etapas [47]: (1) detecção; (2) diagnóstico; e (3) remediação. A etapa de detecção é a fase de um processo de determinação de problemas em que anomalias são visualizadas. A etapa de diagnóstico é a fase de um processo de determinação de problemas em que são identificadas as causas que levaram a anomalia a ocorrer. Por fim, a etapa de remediação é onde as causas que levaram as anomalias a ocorrer são corrigidas.

No contexto deste trabalho, apenas a etapa "detecção" da solução de problemas em sistemas computacionais é abordada e, sendo assim, as etapas "diagnóstico" e "remediação" não fazem parte parte do escopo deste trabalho.

2.1.5 Correlação

A correlação é uma medida estatística utilizada para determinar o quanto uma variável influência o valor de outra variável e vice-versa [28]. Dessa forma, mapeando-se cada métrica de um sistema computacional em uma variável, os métodos de correlação podem capturar os relacionamentos existentes entre as métricas do sistema computacional.

2.2 Monitoramento baseado em correlação

Como mencionado no capítulo anterior, observar individualmente o comportamento de métricas monitoradas não é uma prática suficiente para capturar ou descrever o comportamento de um sistema. É importante observar também possíveis correlações entre essas métricas [17], pois o relacionamento entre elas pode fornecer discernimento sobre o comportamento de sistemas e aplicações. Considere uma aplicação *Web* como exemplo. Se existe uma correlação direta entre o número de requisições HTTP para o servidor *Web* e o número de requisições SQL para o servidor de banco de dados, então podemos esperar que um aumento de requisições no primeiro servidor levará a um aumento de requisições no segundo. Portanto, se existe uma correlação direta ou inversa entre os valores de duas

métricas, podemos prever o valor de uma delas em função da outra. Essa forma de descrever o comportamento de um sistema através de correlações entre métricas monitoradas é denominada monitoramento baseado em correlação [23, 36].

Basicamente, o monitoramento baseado em correlação consiste em coletar dados de métricas que representam o estado do sistema durante períodos de normalidade. Esses dados podem ser então usados para capturar correlações estáveis entre métricas monitoradas usando modelos matemáticos. Os modelos de correlação capturados permitem prever o comportamento ou valor de uma métrica em função do comportamento exibido por outra em um passado recente. Como apenas correlações estáveis são consideradas, os valores previstos por esses modelos tendem a se manter dentro de um intervalo previsível enquanto o sistema opera livre de falhas. No entanto, os valores tendem a se desviar significativamente do intervalo previsto quando ocorre falha ou anomalia no sistema.

O monitoramento baseado em correlação apresenta vantagens importantes. Essa abordagem não exige nenhuma informação de entrada por parte do administrador, pois nenhum conhecimento prévio da estrutura do sistema é requerido. Além disso, essa abordagem é genérica, podendo ser aplicada em qualquer sistema com monitoramento de dados.

A correlação que descreve o relacionamento entre duas variáveis (em nosso caso, duas métricas do sistema) pode ser de vários tipos [17]. Dentre eles, a correlação linear é o modelo mais simples de se calcular, reduzindo, assim, a sobrecarga do sistema de monitoramento. Portanto, consideramos apenas esse tipo de modelo neste trabalho.

Um dos problemas associados ao monitoramento baseado em correlação é a exigência de grande capacidade computacional para a verificação das correlações [36]. Esse inconveniente deu origem aos filtros de correlações.

2.2.1 Filtros de correlações

Os filtros de correlação foram desenvolvidos a fim de reduzir o esforço computacional associado ao monitoramento por meio de correlações [36]. Esses filtros tem como objetivo selecionar as correlações identificadas como mais importantes dentro de um sistema monitorado. Munawar et al. [36] propuseram filtros baseados em aglomeração e na MST, comparando-os com os filtros de seleção aleatória e de seleção por correlação de maior coeficiente. Diferente das técnicas apresentadas nesse trabalho, nossas propostas envolvem a aplicação da correlação parcial na seleção de métricas e correlações. Como mencionado anteriormente, a escolha da correlação parcial baseia-se na obtenção de informações impossíveis de serem obtidas a partir de métodos de correlações comuns.

É importante destacar que como, segundo estudos [50], o número de correlações cresce exponencialmente com o número de métricas coletadas, quanto maior for o

número de componentes dentro de um sistema computacional, mais imprescindível será a utilização de filtros de correlações na seleção de métricas utilizadas no monitoramento.

Na seção seguinte, descrevemos uma forma de mensurar a correlação linear entre duas métricas usando o coeficiente de Pearson. No restante deste trabalho, usamos a seguinte notação. Empregamos letra maiúscula para denotar variáveis aleatórias e usamos letra minúscula para representar valores ou instâncias dessas variáreis.

2.3 Coeficiente de Pearson

Assumindo cada métrica monitorada como uma variável aleatória, uma maneira de mensurar a correlação entre duas métricas X e Y é calcular o coeficiente de correlação de Pearson. O coeficiente de Pearson é definido como a razão entre a covariância e os desvios padrão. Dadas n observações de X e Y, a covariância é dada por:

$$\mathbb{C}oV(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}), \tag{2-1}$$

onde $\bar{x}$ e $\bar{y}$ são, respectivamente, as médias das observações de X e Y. A partir da Equação 2-1, podemos notar que valores de X e Y maiores que suas respectivas médias contribuem para aumentar o valor da covariância. De maneira análoga, valores de X e Y menores que suas respectivas médias também aumentam o valor da covariância. Por outro lado, quando um valor é maior que a média de sua variável aleatória e o outro valor é menor, a covariância diminui. Portanto, se X e Y se comportam da mesma maneira em relação às suas médias, a covariância será positiva. Se X e Y se comportam de maneira oposta, porém de forma consistente, a covariância será negativa. Se elas se comportam de forma inconsistente, ou seja, às vezes se comportam de forma semelhante, às vezes não, a covariância será próxima de zero.

Dividindo a covariância pelo desvio padrão de X (s_x) e Y (s_y), normalizamos seu valor para um número dentro do intervalo [-1,1] e obtemos o coeficiente de Pearson (ρ), como mostrado na Equação 2-2:

$$\rho = \frac{1}{n-1}\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{s_x s_y}, \tag{2-2}$$

onde s_x e s_y são definidos como:

$$s_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})}, \tag{2-3}$$

$$s_y = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i-\bar{y})}. \tag{2-4}$$

O coeficiente de Pearson mede o quanto as métricas X e Y estão linearmente relacionadas. Um coeficiente com valor igual a 1 indica um relacionamento linear direto, enquanto o valor -1 determina um relacionamento linear inverso. O valor 0 indica que não existe correlação linear entre as métricas. Para os demais valores, quanto mais próximo de 0 for o valor absoluto do coeficiente, mais fraca será a correlação linear entre as duas métricas. Analogamente, quanto mais próximo de 1 (ou -1, no caso das correlações negativas), mais forte será a correlação.

2.4 Coeficiente de correlação parcial

O coeficiente de correlação de Pearson é capaz de determinar o quanto duas métricas possuem variações semelhantes. No entanto, o coeficiente de Pearson não provê nenhuma informação sobre a existência de uma terceira variável que influencie essa similaridade. Essa informação pode ser obtida usando o conceito de correlação parcial [2].

A correlação parcial permite investigar o quanto a correlação entre duas variáveis é resultado da correlação de ambas com uma terceira variável mediadora. Por exemplo, no monitoramento de um sistema computacional, suponha que A, B e C são três métricas em que identificamos uma forte correlação entre os pares $A-B$, $B-C$ e $A-C$. Para verificar o quanto a correlação entre o par $A-B$ é resultado das correlações individuais de A e B com C, os relacionamentos entre os pares $A-C$ e $B-C$ são removidos. Após a influência de C ser removida, a correlação entre o par $A-B$ é recalculada, resultando na correlação parcial de A e B. Se o resultado da correlação parcial for significativamente menor que o da correlação original, então a correlação original existia, em sua maior parte, devido à correlação de A e B com C.

O uso de correlação parcial para investigar o comportamento de sistemas complexos tem se tornado popular. Esse conceito já foi aplicado no estudo de redes de genes [8, 15, 39], redes de neurônios [18] e, mais recentemente, redes de mercados financeiros [25, 43]. Nesses estudos, a correlação parcial não é utilizada como uma métrica de causalidade, pois correlações não podem ser interpretadas com esse fim [25]. Entretanto, a correlação parcial é aplicada para eliminar muitas correlações que não representam relacionamentos reais. Em redes ou sistemas complexos, isso é útil pois reduz o conjunto de correlações a ser analisado. Até onde sabemos, este é o primeiro trabalho que usa correlação parcial para estudar redes de métricas monitoradas em um sistema computacional.

A correlação parcial é expressa pelo coeficiente de correlação parcial. Esse coeficiente quantifica a correlação entre duas variáveis quando condicionada a uma terceira. Formalmente, sejam X e Y duas variáveis aleatórias quaisquer. O coeficiente de correlação parcial entre X e Y, dada uma variável aleatória Z, denotado por $\rho(X,Y:Z)$,

pode ser expresso em termos dos coeficientes de Pearson $\rho(X,Y)$, $\rho(X,Z)$ e $\rho(Y,Z)$ como mostrado na Equação 2-5:

$$\rho(X,Y:Z) = \frac{\rho(X,Y) - \rho(X,Z)\rho(Y,Z)}{\sqrt{[1-\rho^2(X,Z)][1-\rho^2(Y,Z)]}}. \quad (2\text{-}5)$$

Na Equação 2-5, um valor alto de $\rho(X,Y:Z)$ indica que Z exerce pouca ou nenhuma influência na correlação entre X e Y. Isso ocorre porque o coeficiente de correlação parcial assume um valor alto apenas se $\rho(X,Y)$ for muito maior que $\rho(X,Z).\rho(Y,Z)$. Por outro lado, um valor pequeno de $\rho(X,Y:Z)$ pode indicar duas situações: (1) Z tem uma forte influência na correlação entre X e Y, ou seja, $\rho(X,Y) \sim \rho(X,Z).\rho(Y,Z)$, (2) a correlação entre as três variáveis é pequena, ou seja, os coeficientes de Pearson $\rho(X,Y)$, $\rho(X,Z)$ e $\rho(Y,Z)$ são pequenos. Para diferenciar entre essas duas situações, usamos uma métrica denominada influência de correlação [25] definida na Equação 2-6:

$$d(X,Y:Z) \equiv \rho(X,Y) - \rho(X,Y:Z). \quad (2\text{-}6)$$

A influência de correlação quantifica a influência de Z no relacionamento das variáveis X e Y. Esse valor é alto apenas quando uma parte significativa da correlação entre X e Y é explicada em função da variável Z. A PCTN, rede apresentada no próximo capítulo e que serve de base para os filtros de correlação propostos neste trabalho, utiliza o conceito de influência de correlação em sua construção.

Na próxima seção, apresentamos uma revisão do estado da arte em detecção de anomalias voltada especificamente para aplicações *Web*. Apresentamos também as principais diferenças entre esses trabalhos e nossa proposta.

2.5 Trabalhos relacionados

Atualmente, muitos centros de dados que executam aplicações *Web* utilizam sistemas de gerenciamento comerciais como o SmartCloud da IBM [21] e o System Center Operations Manager da Microsoft [30] para detectar anomalias. Nesses sistemas, os administradores configuram manualmente regras que estabelecem limiares para um conjunto de métricas monitoradas. Quando o valor de uma dessas métricas extrapola seu limiar, uma anomalia é detectada e um alarme é emitido para o administrador. Além de ferramentas comerciais, alguns trabalhos da literatura também utilizam regras baseadas em limiares. Por exemplo, em [24] é apresentado um método para classificação de limiares extrapolados. Os autores analisam não apenas a regra e seu próprio limiar, mas também limiares equivalentes mapeados a partir de outras métricas.

A detecção de anomalias através de definições de regras apresenta várias desvantagens. Muitas vezes, a configuração dos limiares é realizada de forma manual. Essa configuração não é uma tarefa trivial, pois valores muito altos podem ocasionar um número elevado de falso-negativos, enquanto limiares muito baixos podem incorrer em muitos falso-positivos. Além disso, abordagens baseadas em regras com configuração manual tendem a monitorar um conjunto pequeno de métricas para facilitar essa configuração. Como mencionado, essa decisão pode fazer com que várias anomalias não sejam percebidas pelo mecanismo. Outra dificuldade dessa abordagem é a análise das métricas monitoradas, a qual é realizada de forma isolada para cada métrica. Aplicações *Web*, no entanto, apresentam interações complexas entre seus componentes e o ambiente de execução. Esse comportamento, por outro lado, dificilmente é capturado através de uma métrica isolada.

Jiang et al. [23] foram um dos primeiros a propor a utilização de modelos matemáticos para descrever a relação entre diferentes métricas de uma aplicação *Web*. Nesse trabalho, os autores usam regressão linear para capturar o relacionamento entre os pares de métricas. Assumindo que esse comportamento é invariante ao longo do tempo em situação de normalidade, qualquer mensuração que rompa ou viole esse relacionamento é uma indicação de uma anomalia. Um problema nessa abordagem é determinar quais relações lineares são verdadeiras, uma vez que os modelos são gerados para todas as combinações possíveis envolvendo duas métricas. Para resolver esse problema, os autores estabelecem um fator de confiança para os modelos, o qual é medido periodicamente a partir do número de previsões corretas feitas por esses modelos. Quando o fator de confiança de um determinado modelo cai abaixo de um limiar, ele é removido da solução. Do ponto de vista do monitoramento, os autores não propõem nenhuma forma de seleção de métricas. Essas vão sendo eliminadas da solução de monitoramento à medida que os modelos que as utilizam são eliminados. Um problema dessa abordagem é sua convergência lenta para o conjunto confiável de métricas.

Urgaonkar et al. [46] foram um dos primeiros a tentar caracterizar a carga de trabalho de aplicações *Web*. Esse trabalho usa redes de filas abertas e fechadas e um algoritmo de Análise do Valor Médio – *Mean Value Analysis* (MVA) – para caracterizar o tempo de serviço médio dos servidores de cada camada. Desvios nos tempos de serviço dos servidores indicam a ocorrência de uma anomalia. Uma limitação desse trabalho é a consideração de que a carga é composta por um conjunto de requisições fixas. Uma abordagem mais realista e que leva em consideração cargas compostas por diferentes conjuntos de requisições é apresentada por Cherkasova et al. [10] e por Zhang et al. [52]. Ao contrário do proposto em [46], Cherkasova et al. [10] e Zhang et al. [52] não estimam o tempo médio de serviço de cada servidor, mas, caracterizam o custo individual de cada tipo de transação que pode compor a carga. Para isso, os autores utilizam métodos de regressões estatísticas. Do ponto de vista do monitoramento, todos

esses trabalhos focam apenas a caracterização do padrão de utilização de CPU. Dessa forma, os mecanismos propostos nesses trabalhos não conseguem detectar anomalias de desempenho que envolvam outros recursos computacionais.

Considerando ainda trabalhos a respeito de detecção de anomalias de desempenho em aplicações *Web*, Cohen et al. [12] usam uma rede bayesiana compacta, denominada TAN, para identificar quais métricas se correlacionam com uma violação de nível de serviço. Diferentemente da proposta do presente trabalho, que é aplicar filtros que realizem a seleção de métricas e correlações a serem monitoradas, os autores optaram por utilizar apenas um conjunto pequeno de métricas de sistema para realizar a detecção de anomalias.

Os trabalhos mencionados até o momento, buscam caracterizar o comportamento de aplicações *Web* através de algoritmos que analisam os dados coletados de forma *offline*. No entanto, em ambientes onde a carga de trabalho é dinâmica, essas abordagens tendem a ser pouco eficientes. Recentemente, Wang et al. [48] apresentaram um método para treinamento de padrões de carga de aplicações *Web* que opera de forma *online*. O mecanismo de detecção de anomalias apresentado pelos autores usa um método de aglomeração incremental. Do ponto de vista do monitoramento, os autores não descrevem como as 10 métricas usadas no monitoramento foram selecionadas. No entanto, essas informações têm uma importância significativa em algoritmos de treinamento *online*, nos quais o tamanho da entrada de dados é crítica para o tempo de resposta do mecanismo.

Identificamos que Munawar et al. [36] desenvolveram o trabalho que abordou de maneira mais detalhada o problema de selecionar métricas para mecanismos de detecção de anomalias e, portanto, é o trabalho que mais se assemelha com a nossa proposta. Munawar et al. [36] investigam filtros que são aplicados com o objetivo de diminuir o número de métricas em grafos de monitoramento. Quatro técnicas de filtragem são investigadas: (1) seleção aleatória, (2) seleção por maior correlação, (3) seleção por aglomeração e (4) seleção por MST. De todos os filtros investigados, a MST foi a que obteve melhores resultados. Por esse motivo, no Capítulo 4, essa técnica é comparada com a nossa proposta, a qual se baseia em um filtro construído a partir do coeficiente de correlação parcial.

2.6 Considerações finais

Neste capítulo, foram apresentados os principais conceitos da área de detecção de anomalias em sistemas computacionais e alguns dos trabalhos relacionados mais importantes. Por meio desses trabalhos, vimos que o monitoramento de sistemas computacionais por meio de correlações tem se mostrado uma abordagem promissora, possuindo

vantagens sobre métodos tradicionais que não são capazes de capturar as interações entre as métricas do sistema computacional.

No capítulo seguinte, apresentaremos em detalhe a nossa proposta, incluindo os filtros baseados em correlação parcial e os modulos de software implementadosos para a avaliação desses filtros.

CAPÍTULO 3

Filtros de métricas baseados em correlação parcial

Neste capítulo, apresentamos nossa proposta para selecionar métricas monitoradas em mecanismos de detecção de anomalias que atuam sobre aplicações *Web*. Como mencionado anteriormente, nossa proposta consiste em aplicar filtros baseados em coeficiente de correlação parcial na seleção das métricas e correlações a serem utilizadas por mecanismos para o monitoramento de aplicações *Web*.

3.1 Proposta

A correlação parcial pode ser descrita como um filtro que seleciona uma informação estatística robusta sobre a influência de variáveis específicas na estrutura de correlação de um sistema. Portanto, com a correlação parcial, podemos construir uma rede de correlações que descreve de forma simplificada, porém robusta, as correlações de um sistema complexo. Partindo desse princípio, nossa proposta consiste em construir filtros de métricas para mecanismos de monitoramento baseado em correlação, onde as relações a serem monitoradas são identificadas através de uma rede de correlação parcial. A Figura 3.1 apresenta um funcionamento típico de como mecanismos de monitoramento baseados em correlação são utilizados no monitoramento. Inicialmente, estatísticas sobre métricas monitoradas são coletadas dos diversos componentes do sistema, incluindo os servidores *Web*, de aplicação e de banco de dados, além de métricas do sistema operacional e da rede. Essas estatísticas são coletadas com o intuito de capturar o comportamento normal do sistema. Em seguida, as estatísticas coletadas são correlacionadas entre si, gerando uma correlação para cada par de métricas considerado. As correlações mais importantes do sistema são então capturadas na etapa "filtro de métricas". Em seguida, para cada correlação selecionada, um modelo matemático é derivado para descrever o relacionamento entre o par de métricas pertencentes à correlação. Essa relação é então tomada como uma invariante, ou seja, uma relação que deve se manter válida enquanto o sistema opera livre de falhas. As invariantes são usadas pelo monitor para detectar anomalias no

sistema. Qualquer violação às relações descritas pelas invariantes é uma indicação de uma anomalia.

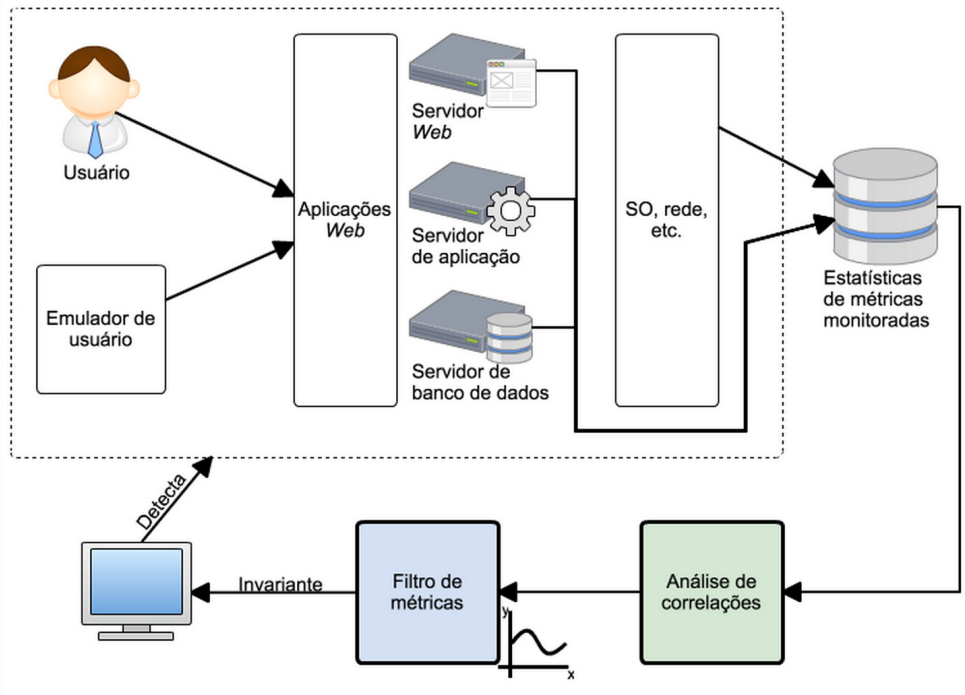


Figura 3.1: *Visão geral do mecanismo de monitoramento.*

Nossa proposta aborda a seleção de métricas realizada na etapa "filtro de métricas", onde propomos, para essa tarefa, a utilização de filtros baseados na correlação parcial. A Figura 3.2 apresenta os seis módulos de software que desenvolvemos para implementar um mecanismo de detecção de anomalias utilizado para validar os filtros baseados em correlação parcial. A seguir, descrevemos brevemente cada um dos módulos.

- Pré-processamento: realiza a preparação dos dados coletados, padronizando e formatando esses dados para que eles possam ser manipulados pelos módulos seguintes.
- Cálculo de correlações: calcula o coeficiente de correlação de Pearson para cada par de métricas e o coeficiente de correlação parcial para cada conjunto de três métricas.
- Filtro de correlações: aplica limiares nas correlações calculadas no módulo anterior para realizar a pré-seleção das métricas e correlações que serão utilizadas na construção dos modelos matemáticos. Nesse módulo implementamos os três filtros baseados em correlação parcial e duas abordagens de filtragem de métricas já existentes na literatura.
- Geração de modelos matemáticos: aplica o método de mínimos quadrados para capturar o relacionamento que descreve cada correlação selecionada pelo módulo anterior.

- Filtro de modelos matemáticos: verifica os modelos matemáticos gerados, retornando apenas os modelos estáveis, também conhecidos como invariantes.
- Detecção de anomalias: utiliza as invariantes para a detecção de anomalias.

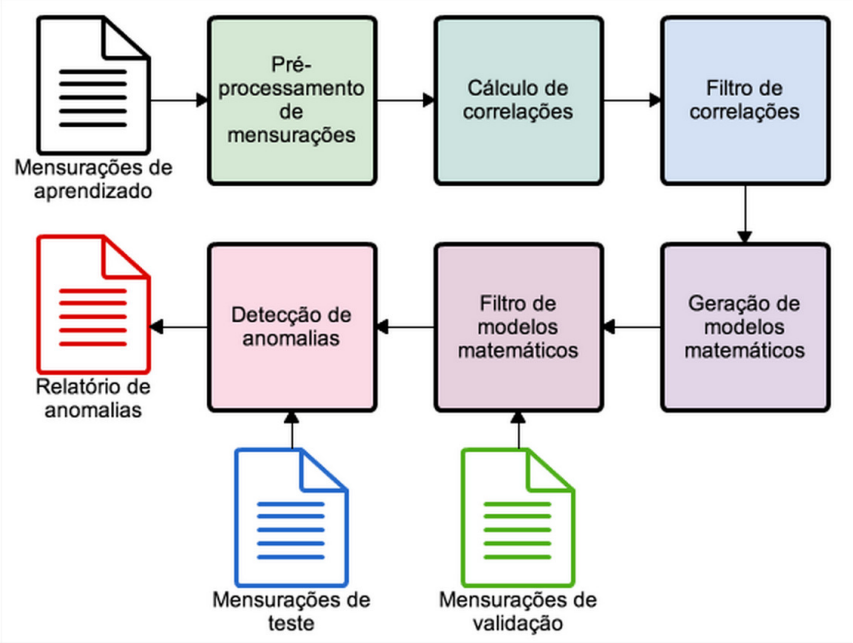


Figura 3.2: *Módulos de software constituintes do mecanismo desenvolvido.*

Vale ressaltar que nossa contribuição de fato se encontra basicamente dentro do módulo de filtro de correlações e que, dessa forma, os outros módulos foram implementados apenas para que resultados pudessem ser extraídos de nossas propostas.

Também é importante mencionar que dentre os seis módulos que formam nosso mecanismo de detecção de anomalias, consideramos que apenas o último (Detecção de anomalias) pode ser executado de forma *online*. Essa consideração foi baseada no fato dos algoritmos de cálculos de correlações e filtros, principalmente os baseados na correlação parcial, exigirem um tempo de processamento elevado.

Nas seções seguintes, descrevemos como as etapas de filtragem de correlações, construção de modelos, verificação de modelos e detecção de anomalias foram implementadas.

3.1.1 Filtro de correlações

Nesta etapa, construímos as redes de correlação parcial que descrevem a estrutura de correlação do sistema. Para construir essas redes, utilizamos o algoritmo proposto por Kenett et al. [25] e denominado *Partial Correlation Threshold Network* (PCTN). Esse algoritmo cria uma rede composta pelas influências de correlação $d(X,Y:Z)$ com valores maiores que um certo limiar. A PCTN é uma rede representada por um grafo, onde

os vértices desse grafo são as métricas monitoradas, enquanto as arestas correspondem às correlações entre as métricas. Dadas as métricas X, Y e Z, acrescentamos uma aresta entre Z e X e outra aresta entre Z e Y, indicando a influência de Z nas métricas X e Y, se, e somente se, elas ainda não existem na rede e a equação a seguir for satisfeita:

$$d(X,Y:Z) \geq \langle d(X,Y:Z)\rangle_Z + k\sigma_Z(d(X,Y:Z)), \tag{3-1}$$

onde $\langle d(X,Y:Z)\rangle_Z$ e $\sigma_Z(d(X,Y:Z))$ são, respectivamente, a média e o desvio padrão determinados com respeito à métrica condicionante Z. O parâmetro k, denominado limiar de influência, define a intensidade de atuação do limiar.

A topologia da rede a ser construída depende fortemente de k. Valores altos de k podem levar a um grafo pouco conexo, o que prejudica a capacidade de detecção do mecanismo. Isso ocorre porque cada aresta representa uma correlação a ser monitorada. Se existem poucas arestas no grafo, o mecanismo irá monitorar poucas correlações e alguns relacionamentos importantes podem não ser representados. Por outro lado, valores muito pequenos de k podem aproximar a topologia da rede de um grafo completo, diminuindo a capacidade de filtragem da rede. Assim como proposto por Kenett et al. [25], este trabalho utiliza uma abordagem empírica para a escolha de k. Entretanto, a abordagem aqui realizada difere da adotada em [25]. Essa divergência ocorreu pois os ambientes de aplicação são diferentes e, com isso, a metodologia original não propiciou discernimento suficiente para a escolha de k. No Capítulo 4, apresentamos em detalhes o motivo que nos levou a abandonar a abordagem empírica original. Além disso, apresentamos também o procedimento que utilizamos para a escolha de k.

Duas abordagens já existentes na literatura também foram implementadas a fim de fornecer informações comparativas nos resultados. A primeira foi a filtragem por valores de coeficiente de correlação de Pearson. A segunda é baseada na utilização do algoritmo MST. Mais detalhes sobre a construção dessas redes serão apresentadas no Capítulo 4.

Como mencionado anteriormente, três redes de monitoramento baseadas em correlação parcial foram construídas, sendo que todas elas utilizam o algoritmo da PCTN. A primeira utilizou todos as métricas coletadas para sua construção, sendo nomeada de rede PCTN. Como nós percebemos características de filtragem muito boas na rede baseada em MST, a segunda e a terceira redes de monitoramento são uma mesclagem entre a PCTN e a MST, formando as redes PCTN-MST e MST-PCTN. A rede PCTN-MST é construída aplicando-se o algoritmo da MST sobre a rede de monitoramento obtida pela PCTN. A rede MST-PCTN é construída aplicando-se o algoritmo da PCTN sobre as métricas selecionadas pela rede de monitoramento MST. Também serão fornecidos mais detalhes sobre essas três redes de monitoramento no Capítulo 4.

É importante destacar que em sua proposta original [25], a PCTN é uma rede re-

presentada por um grafo direcionado. No entanto, em nossa implementação, optamos por um grafo não direcionado pois o foco do nosso mecanismo não é apontar os componentes mais influentes do sistema computacional, mas selecionar os pares de métricas a serem utilizados no monitoramento.

Por fim, também é importante notar que cada uma das redes de monitoramento construídas nesse módulo são avaliadas separadamente pelos próximos módulos e, sendo assim, elas não possuem influências entre si, ou seja, não existe nenhuma colaboração entre as cinco redes construídas nesse módulo.

3.1.2 Construção de modelos

Nesta etapa, consideramos a existência de um relacionamento entre os pares de variáveis X e Y sempre que esse par for selecionado por alguma das redes na etapa anterior. Para cada uma das redes de correlação, esse módulo associa, à cada correlação existente na rede, um modelo matemático linear do tipo $Y = a.X + b$ a fim de descrever esse relacionamento. Para a construção desses modelos matemáticos, utilizamos uma técnica de regressão linear baseada em mínimos quadrados [35].

Dado um conjunto de n amostras, cada uma na forma de um par (x_i, y_i), a regressão linear é uma técnica que encontra os melhores valores de a e b para os exemplos dados. No método dos mínimos quadrados, os melhores valores para a e b são encontrados minimizando uma variável Q que representa a soma dos quadrados dos desvios entre os dados reais e os estimados pelo modelo, conforme descrito pela Equação 3-2:

$$Q = \sum_{i=1}^{n} [y_i - (a.x_i + b)]^2. \tag{3-2}$$

Para encontrar os valores de a e b que minimizam Q, definimos resíduos para cada amostra (x_i, y_i), como mostra a Equação 3-3.

$$r_i = y_i - (a * x_i + b). \tag{3-3}$$

Dessa forma, a Equação 3-2 pode ser reescrita como:

$$Q = \sum_{i=1}^{n} (r_i)^2. \tag{3-4}$$

Além disso, definimos as seguintes matrizes:

$$X = \begin{pmatrix} b \\ a \end{pmatrix}, Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, R = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{pmatrix}, A = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}.$$

Segue que $y_i = a * x_i + b$, para todo i variando de 1 até n, é o mesmo que $AX = Y$. Escrevendo a Equação 3-4 em notação matricial temos:

$$\sum_{i=1}^{n} (r_i)^2 = R^T R, \tag{3-5}$$

onde:

$$R = Y - AX. \tag{3-6}$$

Com isso, o objetivo se torna minimizar a Equação 3-7:

$$Q = R^T R = (Y - AX)^T (Y - AX) = Y^T Y - X^T A^T Y - Y^T AX + X^T A^T AX. \tag{3-7}$$

Para minimizarmos essa equação, o gradiente de Q (derivada primeira de duas variáveis Q) deve ser nulo, ou seja, igual a zero. O gradiente de Q é apresentado na Equação 3-8:

$$\nabla Q = -A^T Y - Y^T A + 2A^T AX = 0 \Rightarrow A^T AX = A^T Y. \tag{3-8}$$

Desse modo, para obtermos os valores de a e b que minimizam a soma dos quadrados dos resíduos, basta resolver o sistema linear:

$$A^T AX = A^T Y. \tag{3-9}$$

Efetuando os cálculos de $A^T A$ e $A^T Y$ temos:

$$A^T A = \begin{pmatrix} \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{pmatrix}, A^T Y = \begin{pmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{pmatrix}. \tag{3-10}$$

Substituindo 3-10 em 3-9, obtemos o sistema linear descrito pela Equação 3-11. Resolvendo esse sistema linear, encontramos os valores de a e b que minimizam a soma dos quadrados dos resíduos.

$$\begin{pmatrix} \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{pmatrix} \tag{3-11}$$

3.1.3 Verificação de modelos

Cada modelo matemático construído no passo anterior é verificado com o intuito de avaliar sua estabilidade. Para realizar essa avaliação, inicialmente, para cada modelo,

construímos um limite de aceitação de resíduos, conforme descrito a seguir.

Seja (x_i, y_i) um exemplo observado para um par de métricas X e Y. Seja $\hat{y}_i$ o valor previsto pelo modelo matemático para a métrica Y, dado o valor x_i para X. O termo resíduo denota a diferença entre o valor realmente observado para a métrica Y e o valor previsto pelo modelo matemático, ou seja:

$$R_i = y_i - \hat{y}_i. \tag{3-12}$$

Seja R(t) o conjunto de todos os resíduos observados para a métrica Y em um período de tempo. Para determinar os limiares superior e inferior do limite de aceitação de resíduos, utilizamos a mesma abordagem empregada em Jiang et al. [23]. Essa abordagem consiste em encontrar um valor $\hat{R}$ que é maior que $99,5\%$ dos resíduos observados ($R(t)$). Em seguida, esse valor é amplificado em 10%, como descrito na Equação 3-13:

$$\tau = \pm 1,1 \times arg_{\hat{R}}\{prob(|R(t)| \leq \hat{R}) = 0.995\}. \tag{3-13}$$

Na Equação 3-13, os valores positivo e negativo do limiar τ são usados, respectivamente, como limiares superior e inferior do limite de aceitação do modelo. Uma vez que esse limite é determinado, o modelo pode ser verificado, como descrevemos a seguir.

A verificação do modelo consiste em submetê-lo a um conjunto de exemplos (x_i, y_i) observados a partir da execução do sistema em um cenário livre de falhas. Cada valor x_i é fornecido como entrada para o modelo, o qual gera o valor $\hat{y}_i$ correspondente. Em seguida, cada valor $\hat{y}_i$ é comparado com o respectivo valor y_i e o resíduo R_i é calculado. O modelo é considerado estável se no máximo uma porcentagem p, denominada Porcentagem de Resíduos Discrepantes Permitidos (PRDP), de observações de resíduos estão fora dos limites de aceitação do modelo.

3.1.4 Detecção de anomalia

A detecção de anomalias ocorre através do monitoramento dos modelos matemáticos considerados estáveis na etapa anterior. Em nossa solução, esses modelos são considerados invariantes, ou seja, os relacionamentos descritos pelos modelos devem persistir enquanto o sistema ou aplicação operar livre de falhas. Periodicamente, esses modelos matemáticos são alimentados com novas observações de métricas e os resíduos desses modelos são recalculados. Se a quantidade de resíduos que violam os limites de aceitação de um modelo for superior à definida pela PRDP, o modelo correspondente é considerado violado e é indicada a ocorrência de uma anomalia.

É importante observar que, em geral, os mecanismos de detecção de anomalias que utilizam monitoramento baseado em correlação não estabelecem como selecionar um conjunto de métricas. Logo, grande parte desses mecanismos iniciam sua execução

a partir do procedimento descrito na Seção 3.1.2, ou seja, criando modelos matemáticos para cada par de métricas considerado. Como consequência, esses mecanismos levam um tempo maior para convergir para um conjunto de modelos estáveis. Nossa proposta, por outro lado, acrescenta o passo descrito na Seção 3.1.1. Esse passo permite selecionar um conjunto de métricas com maior potencial para gerar modelos matemáticos estáveis.

3.2 Considerações finais

Neste capítulo, apresentamos nossa proposta de utilização de filtros de métricas em mecanismos de detecção de anomalias baseados em correlação. Nossa proposta difere da maioria de outros trabalhos pela inclusão de uma etapa de seleção das correlações a serem monitoradas na aplicação *Web*, seleção que é realizada por meio de filtros baseados em correlação parcial. No Capítulo 4, apresentamos uma avaliação experimental dessa proposta e discutimos os resultados obtidos.

CAPÍTULO 4

Avaliação e análise dos resultados

Neste capítulo, avaliamos as redes de monitoramento baseadas na correlação parcial ao ser utilizadas filtros de métricas e correlações em um mecanismo de monitoramento. Os filtros implementados foram avaliados em uma *testbed*, onde foi implantado um ambiente que simula a operação de uma aplicação *Web* multi-camadas. Esse ambiente nos permitiu conduzir os experimentos de forma controlada. Além da nossa proposta, avaliamos outros trabalhos importantes da literatura na mesma *testbed* e sob os mesmos experimentos. Iniciamos este capítulo descrevendo o ambiente de testes implantado para a avaliação experimental.

4.1 Infraestrutura de testes

Nossa *testbed* consiste em uma arquitetura de três camadas que simula a operação de uma livraria *online*. As transações *Web* são geradas através do TPC-W, um *benchmark* clássico que emula comportamentos comuns a *sites* de comércio eletrônico [29].

A Figura 4.1 ilustra uma visão geral da *testbed*. A camada de dados é gerenciada pelo servidor de banco de dados MySQL, instalado no equipamento denominado *Database Server*. A camada de negócio, ou camada lógica, é baseada no servidor de aplicação Apache Tomcat, instalado no equipamento denominado *Front Server*. Nesse equipamento também está instalada a lógica da aplicação *Web*, representada pelo núcleo do TPC-W. Neste trabalho, implantamos uma implementação Java de código aberto do TPC-W, desenvolvida por Bezenek et. al [4]. A camada de apresentação é formada por um módulo de software do TPC-W que emula os clientes da aplicação *Web*. Essa camada é representada na Figura 4.1 pelo equipamento *Client*.

A Tabela 4.1 resume as principais características dos equipamentos que compõem a *testbed*. Todos os equipamentos executam o sistema operacional Ubuntu 12.04 LTS (Precise Pangolin). Adicionalmente, o *Front Server* e o *Database Server* executam também a ferramenta collectd [14], utilizada para realizar a coleta de estatísticas das métricas que serão monitoradas.

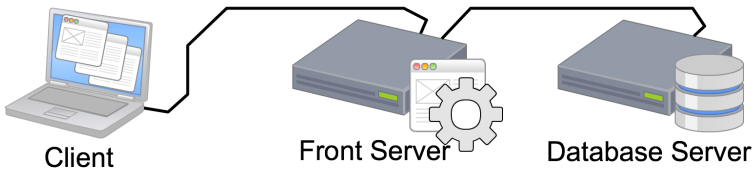


Figura 4.1: *Topologia da testbed.*

Tabela 4.1: *Características dos equipamentos utilizados.*

Equipamento	***Processador***	***Memória RAM***	***Link de rede***	***Software instalado***
Client	Intel(R) Atom(TM) D510 @ 1.66GHz	1 GB	Gigabit	Emulador de clientes do TPC-w
Front Server	Intel(R) Core(TM) i5-3470 @ 3.20GHz	4 GB	Gigabit	Apache Tomcat e collectd
Database Server	Intel(R) Core(TM) i5-3470 @ 3.20GHz	4 GB	Gigabit	MySQL e collectd

Tipicamente, a carga de trabalho de aplicação *Web*, como comércio eletrônico e *sites* empresariais, é melhor descrita no nível de sessão [53]. Uma sessão, nesse contexto, consiste em uma sequência de requisições individuais e consecutivas. Em um *site* de comércio eletrônico, por exemplo, uma sessão convencional envolve as seguintes requisições:

- a seleção de um ou mais produtos através do *site*;
- o fornecimento de informações para entrega do pedido;
- a realização do pagamento através de dados fornecidos pelo cliente e;
- a confirmação da compra.

Todas essas requisições individuais são necessárias para completar a transação de mais alto nível representada por uma venda *online*. Portanto, uma medida de desempenho comumente usada em aplicações *Web* é o número de sessões concorrentes que o sistema pode suportar sem violar um tempo de resposta transacional máximo estabelecido previamente.

De acordo com a especificação do TPC-W, o número de sessões concorrentes, também chamada de navegadores emulados – *Emulated Browser* (EB) –, é constante durante todo o experimento. Para cada EB, o TPC-W define, de forma estatística, o tamanho da sessão do cliente, o tempo de *think* do usuário e as consultas que são geradas

pela sessão. O tamanho médio das sessões é 15 minutos e, sempre que uma sessão termina, uma nova é criada imediatamente. Além disso, o tamanho do banco de dados é determinado pelo número de itens disponíveis para compra e também pelo número de clientes.

Como mostra a Tabela 4.2, o TPC-W define 14 transações diferentes. Além das transações, o TPC-W define três grupos de distribuição de acesso entre essas transações: *Browsing Mix*, *Shopping Mix* e *Ordering Mix*. Nessa tabela, também são mostradas as porcentagens médias de requisições destinadas às transações de cada grupo.

Tabela 4.2: *Distribuição de requisição entre as transações do TPC-W.*

Interação Web	***Browsing Mix***	***Shopping Mix***	***Ordering Mix***
Pesquisa	**95%**	**80%**	**50%**
Home	*29.00%*	*16.00%*	*9.12%*
New Product	*11.00%*	*5.00%*	*0.46%*
Best Seller	*11.00%*	*5.00%*	*0.46%*
Product Detail	*21.00%*	*17.00%*	*12.35%*
Search Request	*12.00%*	*20.00%*	*14.54%*
Search Results	*11.00%*	*17.00%*	*13.08%*
Pedidos	**5%**	**20%**	**50%**
Shopping Cart	*2.00%*	*11.60%*	*13.53%*
Customer Reg.	*0.82%*	*3.00%*	*12.86%*
Buy Request	*0.75%*	*2.60%*	*12.73%*
Buy Confirm	*0.69%*	*1.20%*	*10.18%*
Order Inquiry	*0.30%*	*0.75%*	*0.25%*
Order Display	*0.25%*	*0.66%*	*0.22%*
Admin Request	*0.10%*	*0.10%*	*0.12%*
Admin Confirm	*0.09%*	*0.09%*	*0.11%*

4.2 Metodologia de testes

Nesta seção, descrevemos a metodologia que empregamos para realizar a avaliação experimental. A metodologia adotada compreende três etapas: geração da carga de trabalho, coleta de estatísticas e construção das redes de monitoramento. Essas etapas são descritas nas Seções 4.2.1 e 4.2.2.

4.2.1 Carga de trabalho e coleta de estatísticas

A primeira etapa consiste em gerar a carga de trabalho que será utilizada para construir a rede de correlações. Como essa carga se destina à construção e treinamento

dos modelos que descrevem a relação entre os pares de métricas, ela é aplicada sobre o sistema livre de falhas.

Para a avaliação experimental, optamos pelo grupo de carga *Browsing Mix*. Esse grupo foi escolhido por representar um padrão de acesso fortemente baseado em leitura, como mostra a Tabela 4.2. Como grande parte das transações realizadas em *sites* de comércio eletrônico envolvem pesquisas que não resultam efetivamente em pedido ou compra, esse padrão descreve um cenário comumente encontrado em ambientes de comércio pela *Web*.

O número de EBs utilizados na geração da carga de trabalho foi escolhido de forma experimental. Para definir esse número, executamos um conjunto de experimentos, cada um com 30 minutos de duração, e variamos o número de EBs em 300, 450 e 600. Nos experimentos em que o número de EBs foi igual a 300, não observamos atrasos na coleta de estatísticas sobre a execução da aplicação. Nos experimentos em que esse número foi igual a 450, observamos um atraso maior que $2,5$ segundos na coleta das estatísticas. Nos experimentos com 600 EBs, esse atraso foi de aproximadamente 30 segundos. Esses resultados evidenciam que valores maior que 300 EBs levaram nosso sistema à saturação. Essa constatação está de acordo com outros trabalhos que também analisaram a carga do TPC-W, como Zhang et al. [53] e Wang et al. [48]. Portanto, após a realização desses experimentos, definimos em 300 o número de EBs usados na carga de trabalho *Browsing Mix*.

Utilizando a configuração descrita, executamos ao todo 52 experimentos de 30 minutos e 31 experimentos de 360 minutos (ou 6 horas). Inicialmente, definimos o tempo de execução dos experimentos em 6 horas. Esse tempo foi escolhido por ser um período relativamente longo, o que possibilita a obtenção de modelos mais estáveis. No entanto, por restrições de tempo, fomos obrigados a reduzir o tempo de experimentação para apenas 30 minutos. Além disso, a quantidade de experimentos com 30 minutos foi fixada em 52 para, além de fornecer os dados para a escolha das correlações, construções dos modelos e testes de estabilidade, restar uma quantidade experimentos sem falhas igual à quantidade de experimentos com falhas (descritos na Seção 4.2.3). O objetivo dessa igualdade é evitar distorções na avaliação de acurácia, as quais podem ser causadas pela diferença dessas quantidades. Nas seções a seguir, nos referimos às cargas de trabalho de 30 e 360 minutos como *Browsing_mix_30* e *Browsing_mix_360*, respectivamente.

Com o intuito de capturar o comportamento da aplicação *Web* ao longo do tempo, monitoramos 396 métricas enquanto o sistema era submetido às cargas de trabalho descritas anteriormente. Essas métricas incluem estatísticas sobre utilização de CPU, memória, disco e rede, além de métricas do sistema operacional e da máquina virtual Java (JVM). A lista completa das métricas utilizadas neste trabalho é apresentada no Apêndice A.

A coleta de estatísticas é realizada no final de cada janela de monitoramento. Todas as janelas têm tamanho fixo de 5 segundos. Esse valor foi adotado após observarmos que era pequeno o suficiente para termos uma quantidade adequada de amostras e que o impacto no consumo de CPU era negligenciável, permanecendo abaixo de $0,5\%$.

Um fator importante na coleta desses dados é a sincronização dos relógios dos equipamentos monitorados. Divergências nos horários registrados nesses equipamentos podem inviabilizar o monitoramento por meio de correlações, uma vez que o cálculo das correlações e o monitoramento poderiam ser realizados com mensurações de métricas obtidas com tempos reais muito diferentes. Para a sincronização dos relógios de nossos equipamentos utilizamos o *Network Time Protocol* (NTP). No início de cada geração de carga, os relógios dos equipamentos eram comparados e, se houvesse uma diferença visível entre os relógios, a geração de carga era abortada.

4.2.2 Construção das redes de monitoramento

Após submeter o sistema às cargas *Browsing_mix_30* e *Browsing_mix_360* e coletar estatísticas das 396 métricas monitoradas durante a execução do sistema, formamos uma base de dados usada para criar a rede de correlação parcial. Além dessa rede, para efeito de comparação, outras redes também foram criadas a partir da mesma base de dados. A seguir descrevemos o processo de criação de cada rede.

Rede de Pearson

A rede de Pearson foi criada seguindo os passos seguintes. Utilizando a base de dados gerada pela carga *Browsing_mix_30*, primeiramente calculamos o coeficiente de Pearson para cada par de métricas monitoradas. Em seguida, selecionamos todos os pares que apresentaram correlação forte. Para a rede de Pearson, consideramos que a correlação entre duas métricas é forte se o coeficiente de correlação de Pearson (Equação 2-2) para as duas métricas é maior que 50% [13].

A Figura 4.2 mostra como os valores dos coeficientes de Pearson da carga *Browsing_mix_30* estão distribuídos entre onze intervalos que cobrem os valores de 0% a 100% em passos de 10%. Na figura, as barras vermelhas representam a quantidade de correlações que foram consideradas fracas por estarem abaixo de 50%. Analogamente, as barras verdes representam a quantidade de correlações consideradas fortes, de acordo com o limiar adotado. É possível perceber que a grande maioria das correlações possuem valores muito baixos. Do total de 78.210 correlações, envolvendo as 396 métricas, apenas 1.773 correlações, envolvendo 192 métricas, foram identificadas como fortes.

Para a construção de um grafo representativo da rede de Pearson, as métricas pertencentes às correlações identificadas como fortes foram usadas como vértices do

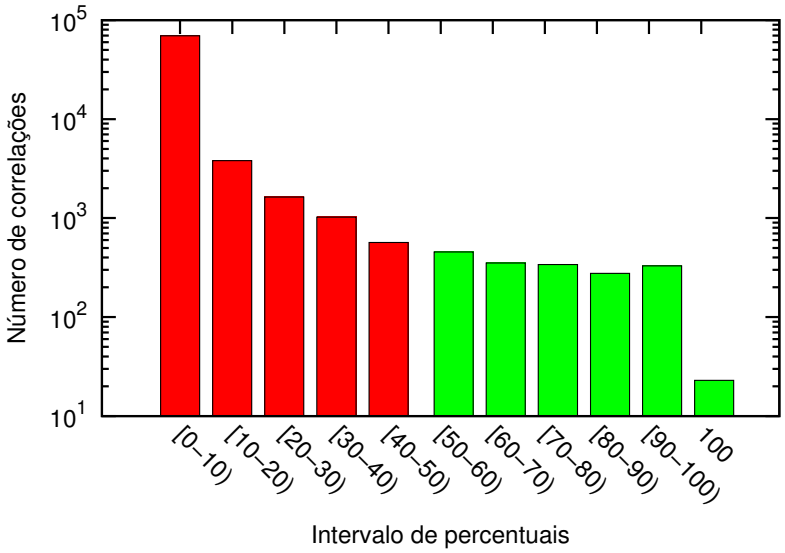


Figura 4.2: *Distribuição dos valores dos coeficiente de Pearson para a carga Browsing_mix_30.*

grafo, enquanto as correlações formaram as arestas. Dessa forma, uma aresta entre dois vértices no grafo indica uma correlação forte entre as duas métricas ligadas. A Figura 4.3 mostra a rede de Pearson formada pela carga de trabalho *Browsing_mix_30* e o limiar de 50% para o coeficiente de Pearson. É possível notar que mesmo removendo muitas correlações, o número de arestas do grafo ainda é elevado. Consequentemente, o monitoramento dessa rede ainda possui alto custo.

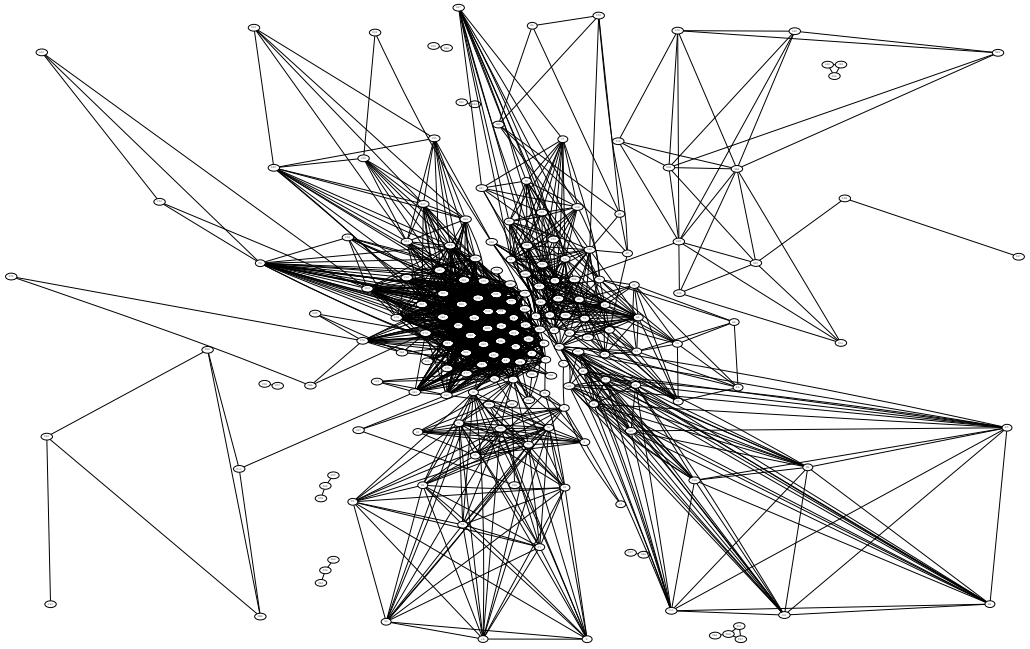


Figura 4.3: *Rede de Pearson para a carga Browsing_mix_30 e limiar de* 50%.

Rede PCTN

A PCTN foi criada seguindo os passos seguintes. Utilizando a base de dados gerada pela carga *Browsing_mix_30*, calculamos a influência de correlação $d(X,Y:Z)$ para cada combinação de métricas X, Y e Z, conforme a Equação 2-6. Em seguida, construímos o grafo da rede, acrescentando uma aresta entre Z e X e entre Z e Y, em todas as combinações em que a influência de correlação satisfaça a Equação 3-1.

Como mencionado no Capítulo 4, a Equação 3-1 exige a definição de um limiar de influência k. O valor desse limiar tem impacto direto sobre a topologia e as propriedades da rede PCTN. Em geral, valores pequenos de k resultam em redes com grandes quantidades de arestas. Como cada aresta representa uma correlação, valores pequenos de k implicam em filtros pouco eficientes. Por outro lado, valores grandes de k dão origem a redes pouco conexas. Portanto, o valor adequado de k é aquele que seja suficientemente alto para reduzir de maneira significativa o número de arestas, mas que também seja baixo o suficiente para manter um componente conexo com grau razoavelmente alto na rede.

Kenett et al. [25] adotaram uma abordagem experimental para encontrar o valor mais adequado para k. Os autores variam o valor desse parâmetro de forma iterativa e avaliam, para cada rede formada, a soma dos pesos das arestas da rede e o grau do vértice mais conexo. O procedimento adotado para dar pesos às arestas foi: para cada aresta $Z \rightarrow X$, o peso dessa aresta é igual ao número de variáveis Y que satisfazem a Equação 3-1. Seguindo uma estratégia similar, variamos o valor de k e analisamos as duas informações apresentadas por Kenett et al. [25] sobre as redes PCTN formadas. Além disso, também analisamos a quantidade de influências retornadas pela PCTN. A Figura 4.4 apresenta as três informações: (1) a soma dos pesos de todas as arestas [$E_w(k)$], (2) o grau do vértice de maior grau [$N_{LC}(K)$] e (3) a quantidade de influências retornadas [$I(k)$]. Os valores de $E_w(0)$, $N_{LC}(0)$ e $I(0)$, ou seja, de $E_w(k)$, $N_{LC}(k)$ e $I(k)$ quando $k = 0$, são:

- $E_w(0) \cong 130.000.000$
- $N_{LC}(0) = 216$
- $I(0) = 4.174.324$

Ao contrário do obtido por Kenett et al. [25], não observamos uma redução abrupta no somatório dos pesos das arestas ao mesmo tempo em que o grau do vértice mais conexo se mantivesse relativamente estável. Como podemos ver todas as três informações avaliadas apresentaram comportamentos similares. Em seu estudo, Kenett et al. [25] mostrou que algumas ações do mercado financeiro possuem grande influência sobre uma grande parte de outras ações. Entretanto, no caso dos sistemas computacionais o mesmo comportamento não pôde ser visualizado significando que as influências de

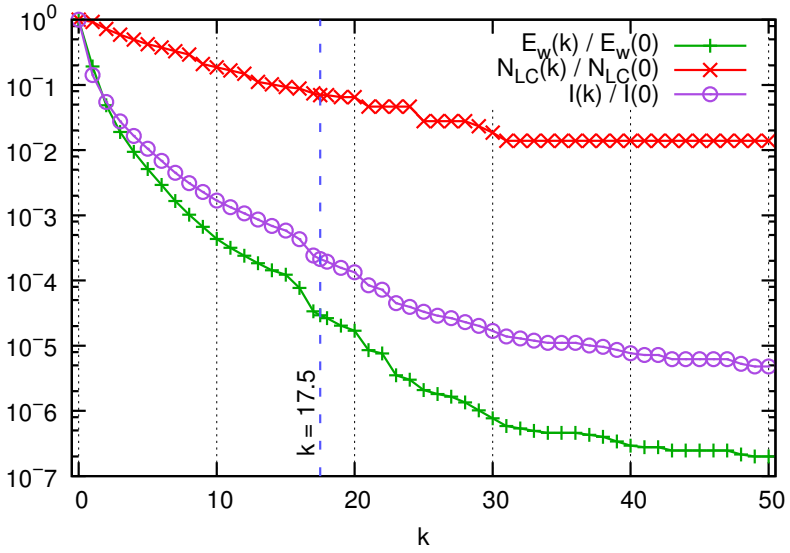


Figura 4.4: *Somatório dos pesos das arestas, grau do vértice mais conexo e número de influências retornadas pela PCTN em função do parâmetro k.*

um sistema computacional tendem a ser mais distribuídas entre suas métricas, não sendo concentradas em poucas métricas como no mercado financeiro.

Diante desse problema, adotamos uma estratégia diferente para determinar o valor de k. Nessa estratégia, variamos iterativamente o valor desse parâmetro até que o número máximo de correlações retornadas pela PCTN fosse igual ao número de correlações existentes na rede de Pearson. Como duas correlações podem ser originadas de cada influência retornada pela Equação 3-1, a saber, uma correlacionando Z e X e a outra Z e Y, o número de influência retornada deve ser igual à metade do número de correlações na rede de Pearson.

Para a carga *Browsing_mix_30*, o número de influências retornado pela rede PCTN é aproximadamente a metade do número de correlações na rede de Pearson quando $k = 17,5$. Usando esse valor como limiar de influência na Equação 3-1, obtivemos uma rede com 396 correlações e 177 métricas. Essa rede é ilustrada na Figura 4.5. Como podemos observar, apesar do número de influências retornadas pela PCTN ter sido a metade do número de correlações da rede de Pearson, o número de correlações na rede PCTN foi bem menor que na primeira rede. Isso ocorreu porque muitas influências retornaram correlações repetidas, sendo essas repetições descartadas.

Rede MST

Essa rede foi proposta por Munawar et al. [36] e consiste em um método para selecionar métricas monitoradas através de árvores geradoras mínimas – MST. Dado um grafo não orientado e conexo, uma árvore geradora mínima é qualquer subgrafo que

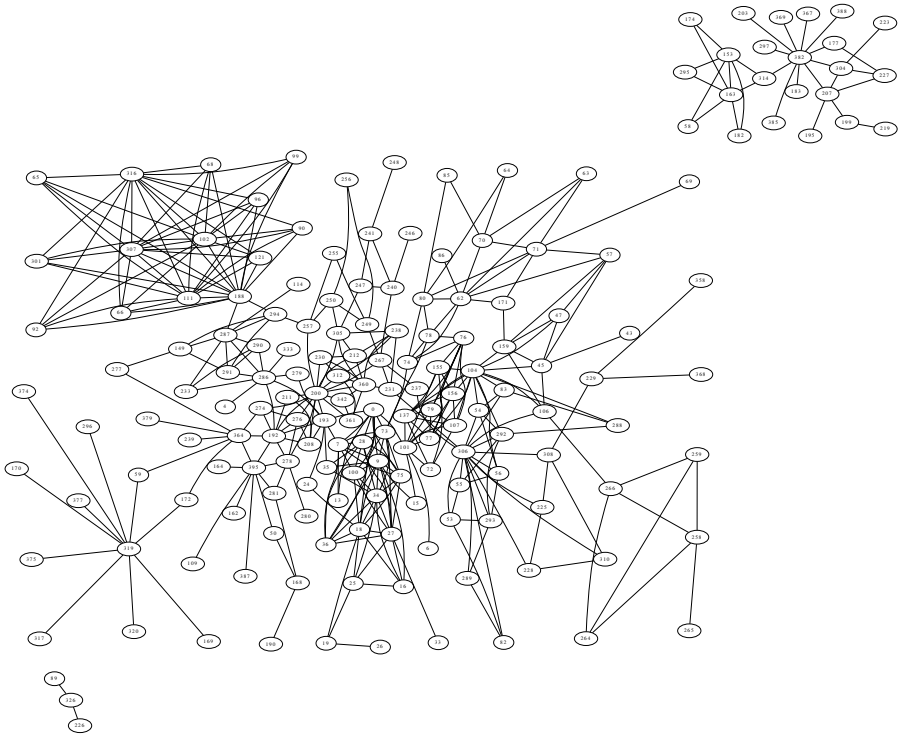


Figura 4.5: *Rede PCTN para a carga Browsing_mix_30 e* $k = 17,5$.

contenha todos os vértices do grafo com o mínimo de arestas possível. Se associarmos um peso ou custo a cada aresta desse grafo, a MST é a árvore geradora com peso ou custo mínimo [40].

Munawar et al. usam o conceito de MST para filtrar as métricas mais importantes em uma rede de Pearson. Dado um conjunto de métricas monitoradas, os autores inicialmente constroem a rede de Pearson para selecionar as correlações fortes. Em seguida, cada aresta da rede é rotulada com um peso igual a $1-\rho(X,Y)$, onde $\rho(X,Y)$ é o valor do coeficiente de Pearson da correlação representada pela aresta. Por fim, a MST é calculada para a rede rotulada. As arestas da MST gerada representam as correlações identificadas como mais importantes da rede.

Com o intuito de avaliar o desempenho da rede PCTN em relação à MST, aplicamos o método proposto por Munawar et al. ao conjunto de dados gerado pela carga *Browsing_mix_30*. Para construir a rede de Pearson, utilizamos o limiar de 50%. A Figura 4.6 apresenta a rede MST formada. Essa rede possui 183 correlações envolvendo 192 métricas. É possível notar que a rede MST possui menos correlações que a PCTN, embora envolva um número maior de métricas.

Redes MST-PCTN e PCTN-MST

Além de avaliar o desempenho da rede PCTN em relação à MST, avaliamos também o desempenho de redes formadas pela combinação dos dois métodos, a saber: a

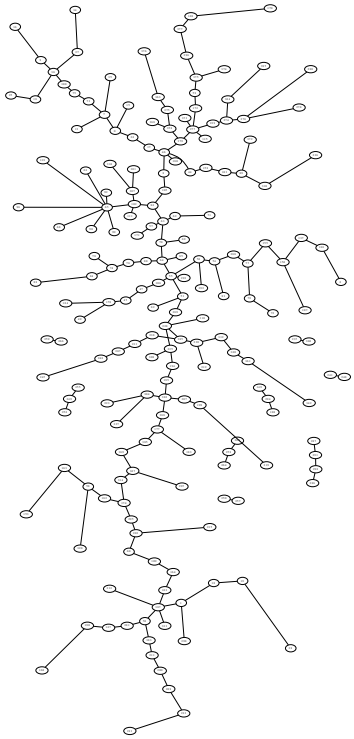


Figura 4.6: *Rede MST para a carga Browsing_mix_30 e limiar de* 50%.

rede MST-PCTN e a PCTN-MST.

A rede MST-PCTN é formada aplicando o filtro baseado em correlação parcial sobre as métricas selecionadas pela rede MST. Para isso, o limiar de influência k da Equação 3-1 foi definido de forma que a PCTN retornasse uma quantidade de influências de correlação igual à metade da quantidade de correlações existentes na rede MST (semelhantemente ao descrito na construção da rede PCTN, porém substituindo os termos "rede de Pearson" por "rede MST"). Isso ocorreu quando o valor de k foi igual a $15,13$. Com isso, o número máximo de correlações selecionadas pela rede MST-PCTN foi limitado pelo número de correlações existentes na rede MST.

Como ocorreu na rede PCTN, muitas influências retornaram correlações repetidas, resultando em uma rede com apenas 62 correlações e 42 métricas. A Figura 4.7 apresenta o grafo da rede MST-PCTN formada.

A rede PCTN-MST, por sua vez, é formada inicialmente através da geração da rede PCTN a partir da base de dados produzida pela carga *Browsing_mix_30* e pela aplicação do limiar de influência $k = 17,5$. Em seguida, o filtro baseado em MST é aplicado sobre a rede formada pela PCTN. A Figura 4.8 apresenta a rede PCTN-MST. A rede resultante possui 174 correlações, envolvendo 177 métricas. Logo, o número de métricas e também de correlações da rede PCTN-MST é menor que os da MST.

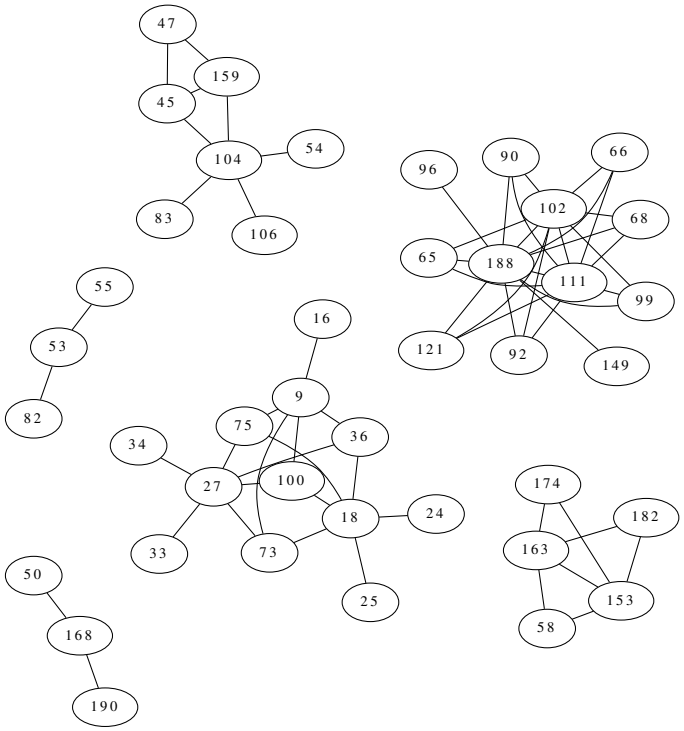


Figura 4.7: *Rede MST-PCTN para a carga Browsing_mix_30, limiar* 50% *e* $k = 15,13$.

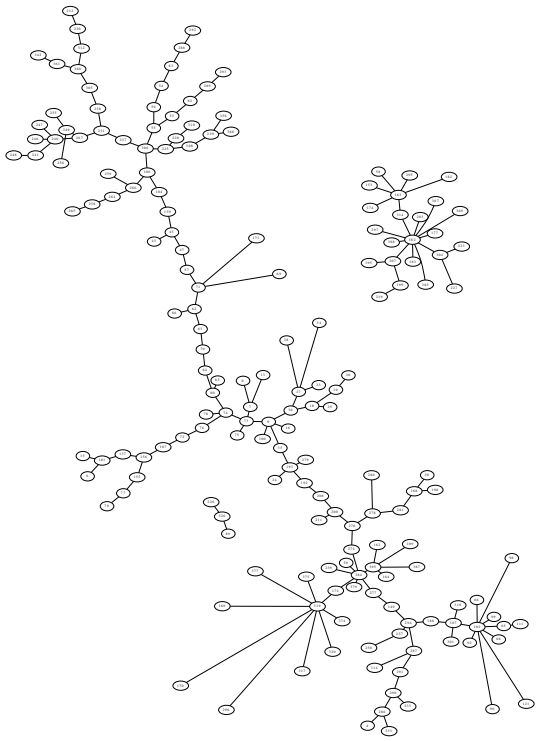


Figura 4.8: *Rede PCTN-MST para a carga Browsing_mix_30 e* $k = 17,5$.

4.2.3 Injeção de falhas

Utilizando a base de dados gerada pela carga *Browsing_mix_30*, construímos as redes de monitoramento descritas na Seção 4.2.2. As correlações selecionadas em cada rede de monitoramento são consideradas invariantes, ou seja, relações matemáticas que devem persistir enquanto o sistema opera livre de falhas. As invariantes são usadas como base para a detecção de falhas.

Para avaliar a capacidade de detecção de falhas das redes de monitoramento, escolhemos algumas falhas típicas de sistemas computacionais [37, 48] e as injetamos em nosso ambiente. As falhas injetadas podem ser classificadas em: falha de software ou de programação, falha de desempenho, falha de configuração e falha de comunicação. A seguir, descrevemos brevemente como essas falhas foram emuladas e injetadas em nosso ambiente.

- Falha de software: escolhemos o *laço de espera ocupada* como a falha de software. Como proposto no trabalho de Jiang et al. [22], inserimos esse laço, com 300 iterações, no início das páginas *Web* do TPC-W listadas na Tabela 4.2.
- Falha de desempenho: para simular uma falha de desempenho no servidor *Web*, inserimos uma espera de 9 segundos no início do carregamento das páginas listadas na Tabela 4.2. Escolhemos o intervalo de 9 segundos com base em estudos que demonstram que, em média, os usuários esperam no máximo 8 segundos pelo carregamento de páginas *Web* antes de desistirem do acesso [16, 49].
- Falha de configuração: para simular uma falha de configuração, reduzimos de 2 GB para 256 MB, a quantidade de memória RAM disponível para uso do servidor Tomcat.
- Falha de comunicação: as falhas de comunicação foram simuladas de duas maneiras. No primeiro cenário, injetamos perdas de pacotes entre o *Front Server* e o *Database Server* com probabilidade de $0,01\%$. As perdas de pacotes foram inseridas utilizando a ferramenta *netem* [27]. No segundo cenário, usamos novamente a ferramenta *netem* para injetar atrasos entre 90 e 110 milissegundos na entrega de pacotes entre o *Front Server* e o *Database Server*.

Realizamos, ao todo, 31 experimentos com injeção de falhas, onde 2 envolveram as falhas de comunicação, 1 envolveu a falha de configuração, 14 envolveram a falha de desempenho e 14 envolveram a falha de software. Vale destacar que os experimentos com falhas de software e de desempenho foram realizados inserindo uma única falha em uma única página *Web* por experimento. Como o TPC-W possui 14 páginas *Web*, esses duas falhas resultaram em 14 experimentos cada uma. Além disso, cada experimento durou 30 minutos e em cada execução apenas uma falha foi injetada. Por fim, com exceção da falha de configuração, todas as falhas foram inseridas em tempo de execução, mais

especificamente na metade do experimento. Devido a falha de configuração exigir o recarregamento da aplicação Tomcat, ela foi inserida antes do início dos experimentos.

4.3 Resultados

Nesta seção, apresentamos os resultados da avaliação das redes baseadas na PCTN em relação às demais redes de monitoramento descritas na Seção 4.2.2. Os experimentos foram conduzidos de forma a avaliar: (1) a estabilidade dos modelos matemáticos gerados pelas redes; (2) a capacidade de detecção de falhas das redes; (3) emissão de falso-positivo ou falso-alarme; e (4) o tempo que as redes levaram na análise das coletas de estatísticas, ou seja, o tempo necessário para apontar falhas se elas existirem nas coletas de estatísticas. Vale ressaltar que parte dos resultados aqui apresentados foram publicados em [38].

4.3.1 Estabilidade dos modelos

Apesar dos modelos matemáticos gerados a partir das redes de monitoramento terem sido criados de correlações identificadas como fortes, uma avaliação importante consiste na estabilidade desses modelos. Consideramos um modelo estável, se as relações invariantes persistem em diferentes execuções do sistema, em um cenário livre de falhas.

Dessa forma, para avaliar a estabilidade dos modelos gerados, primeiramente, construímos a rede PCTN a partir da carga *Browsing_mix_360*. As correlações selecionadas por essa rede foram então usadas para gerar os modelos matemáticos utilizados na detecção de anomalias. Em seguida, submetemos os modelos a testes sem injeção de falhas e observamos a quantidade de modelos que passaram nos testes, ou seja, que reportam estado livre de falhas. Realizamos essa avaliação considerando valores entre 0 e 90% de PRDP (Porcentagem de Resíduos Discrepantes Permitidos). A PRDP indica a porcentagem de resíduos fora dos limites de aceitação que um modelo pode reportar sem que essa situação caracterize uma anomalia ou falha. Além disso, realizamos 10 testes para cada valor de PRDP. Os modelos que falharam em um teste foram eliminados dos testes seguintes.

Na Figura 4.9, apresentamos o comportamento dos modelos gerados pelas redes PCTN para alguns valores de PRDP, a saber: 1%, 2%, 10% e 90%. Para efeito de comparação, apresentamos também o comportamento dos modelos gerados a partir da rede de Pearson, construída sobre as mesmas condições da rede PCTN. Podemos perceber que a filtragem pela PCTN resultou em uma quantidade muito menor de modelos que aquela feita por Pearson. Além de serem em menor quantidade, os modelos derivados da PCTN foram mais estáveis, ou seja, a quantidade de modelos aceitos não variou

significativamente à medida que o número de testes de verificação aumentou. Portanto, esses modelos tem menor probabilidade de emitir falso-positivo, conforme mostraremos posteriormente.

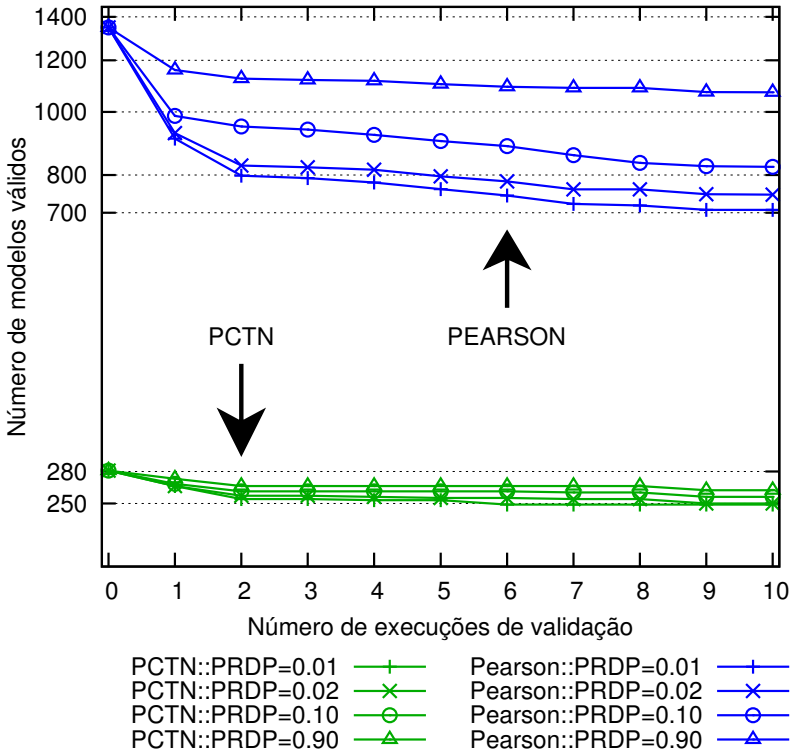


Figura 4.9: *Avaliação da Estabilidade dos modelos gerados pelas redes PCTN e de Pearson para a carga Browsing_mix_360.*

Em seguida, comparamos a estabilidade dos modelos gerados a partir da rede PCTN com a estabilidade dos modelos gerados pelas outras redes de monitoramento. Para essa avaliação, todas as redes de monitoramento foram construídas a partir da carga *Browsing_mix_30*. Como no experimento anterior, as correlações selecionadas por cada rede foram usadas para gerar os modelos matemáticos pertencentes a essas redes. Em seguida, os modelos matemáticos foram submetidos a testes onde nenhuma falha foi injetada. Realizamos essa avaliação considerando 23 valores de PRDPs, os quais variaram da seguinte forma:

- de 0% a 9% com passos de 1%;
- de 10% a 45% com passos de 5%;
- de 50% a 90% com passos de 10%.

Realizamos 20 testes para cada valor de PRDP. As Figuras 4.10(a), 4.10(b), 4.10(c), 4.10(d) e 4.10(e) mostram o comportamento dos modelos em cada rede de monitoramento para os seguintes valores de PRDP: 1%, 2%, 10% e 90%. Como esperado,

podemos notar que quanto menor o valor da PRDP, maior a quantidade de modelos eliminados. Podemos observar também que os modelos matemáticos se mostraram mais estáveis no cenário *Browsing_mix_360* do que no cenário *Browsing_mix_30*. Parte disso se deve ao período de treinamento maior que foi de 6 horas no primeiro cenário contra apenas 30 minutos no segundo cenário. Além disso, analisando os resultados do cenário *Browsing_mix_30*, podemos perceber que a rede de Pearson foi a que mais sofreu descarte de modelos, caracterizando-se como a rede menos estável. As redes MST e PCTN-MST obtiveram uma estabilidade melhor que as redes de Pearson e PCTN. Vale destacar que a MST-PCTN, apesar do pequeno número de correlações, também sofreu um descarte expressivo para valores de PRDP inferiores a 3%.

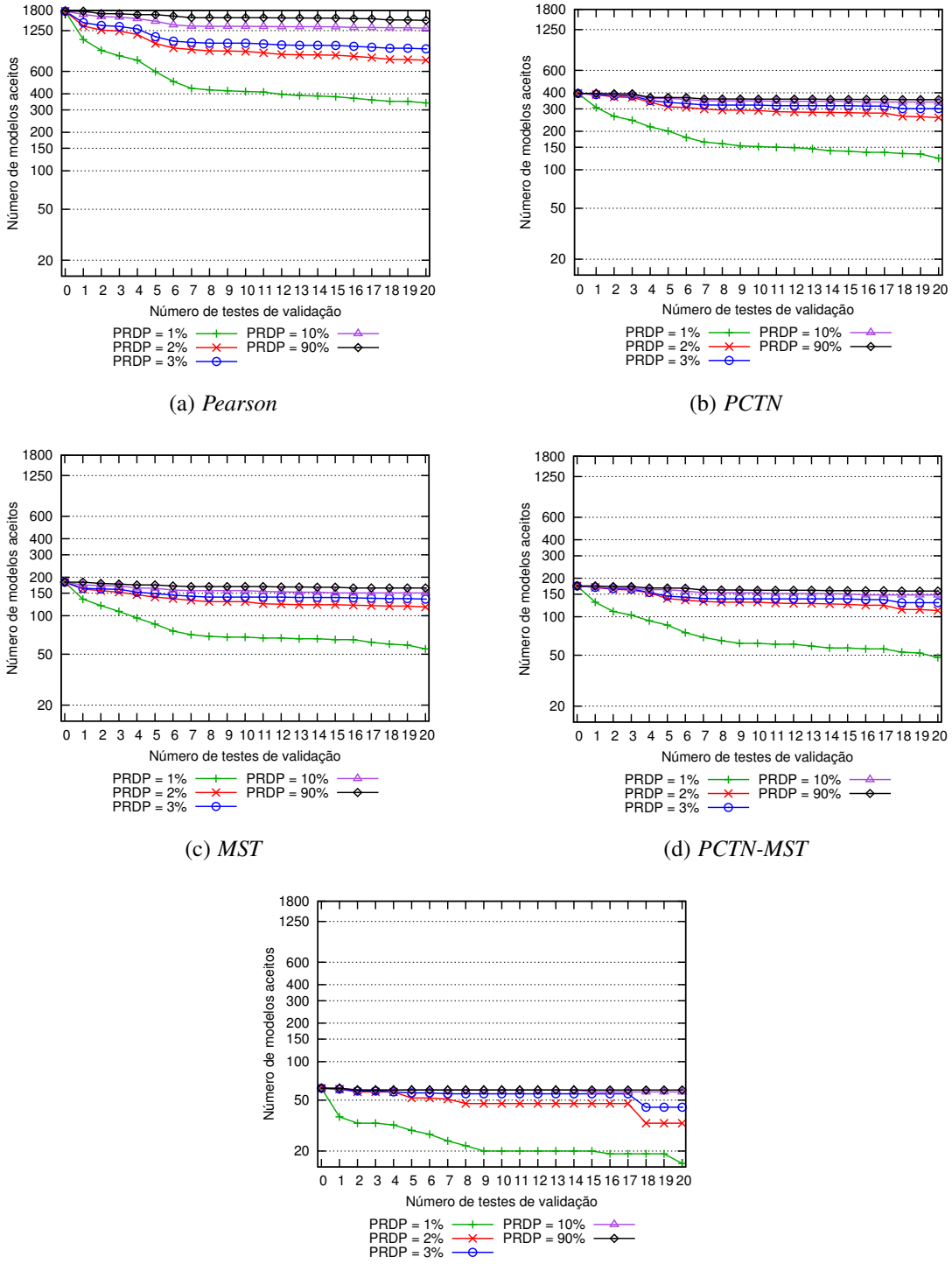
Como mostraremos posteriormente, as redes de monitoramento obtiveram seus melhores resultados dentro do intervalo de PRDP 1% a 3%. Portanto, destacamos na Tabela 4.3 o número de modelos aceitos nesse intervalo. Além disso, na tabela também são apresentadas as quantidades de métricas referenciadas por esses modelos. Ao lado da quantidade de modelos e métricas, entre parênteses, encontram-se as respectivas porcentagens que essas quantidades representam quando comparadas às quantidades das redes originais. Esses modelos aceitos nos experimentos de estabilidade foram utilizados nas demais avaliações: detecção de falhas, falso-positivos e tempo de resposta.

Tabela 4.3: *Estatísticas das redes pós teste de estabilidade.*

Rede de monitoramento	Modelos	Métricas
Pearson - $PRDP = 1\%$	340 (19,18%)	182 (94,79%)
Pearson - $PRDP = 2\%$	735 (41,46%)	184 (95,83%)
Pearson - $PRDP = 3\%$	900 (50,76%)	184 (95,83%)
PCTN - $PRDP = 1\%$	123 (31,06%)	159 (89,83%)
PCTN - $PRDP = 2\%$	256 (64,65%)	161 (90,96%)
PCTN - $PRDP = 3\%$	301 (76,01%)	162 (91,53%)
MST - $PRDP = 1\%$	55 (30,05%)	170 (88,54%)
MST - $PRDP = 2\%$	117 (63,93%)	175 (91,15%)
MST - $PRDP = 3\%$	135 (73,77%)	175 (91,15%)
PCTN-MST - $PRDP = 1\%$	48 (27,59%)	158 (89,27%)
PCTN-MST - $PRDP = 2\%$	112 (64,37%)	160 (90,40%)
PCTN-MST - $PRDP = 3\%$	129 (74,14%)	161 (90,96%)
MST-PCTN - $PRDP = 1\%$	16 (25,81%)	40 (95,24%)
MST-PCTN - $PRDP = 2\%$	33 (53,23%)	41 (97,62%)
MST-PCTN - $PRDP = 3\%$	44 (70,97%)	41 (97,62%)

4.3.2 Detecção de falhas

Para avaliar a capacidade de detecção de falhas das redes de monitoramento, realizamos 31 experimentos com duração de 30 minutos cada. Em cada experimento,



(a) *Pearson*

(b) *PCTN*

(c) *MST*

(d) *PCTN-MST*

(e) *MST-PCTN*

Figura 4.10: *Avaliação da estabilidade dos modelos gerados pelas redes de monitoramento para a carga Browsing_mix_30.*

injetamos uma única falha, e, com excessão do experimento que envolve falha de configuração, todas as demais falhas foram inseridas na metade do tempo do experimento. A seguir, descrevemos os resultados obtidos.

Falha de software

Nesta avaliação, os códigos das páginas *Web* listadas na Tabela 4.2 foram alterados para conter um laço com 300 iterações, como descrito na Seção 4.2.3. Para essa avaliação, realizamos 14 experimentos e, em cada experimento, uma página diferente foi modificada. A Figura 4.11 mostra os resultados dessa avaliação para valores de PRDP variando de 0 a 10%, intervalo onde as redes se mostraram mais acuradas na detecção de falhas. Nessa primeira avaliação, os melhores resultados de todas as redes foram obtidos com a $PRDP = 1\%$. Com essa PRDP, a rede de Pearson e PCTN detectaram todas as falhas de software, enquanto as redes MST e PCTN-MST detectaram mais de 85% das falhas. Com $PRDP = 3\%$ também consideramos os resultados satisfatórios, pois as redes PCTN e PCTN-MST detectaram mais de 80% das falhas, enquanto Pearson se manteve com 100% das falhas detectadas. Um dos motivos que levou Pearson a ter obter esse ótimo resultado foi a maior cobertura de sua rede, monitorando muito mais métricas e correlações que as demais redes. No entanto, essa maior cobertura também resulta em um maior poder de processamento despendido no monitoramento. Vale destacar também a baixa acurácia da rede MST-PCTN que, devido a sua pequena quantidade de métricas e correlações, não detectou nem 30% das falhas em sua melhor parametrização. Dessa forma, nesta avaliação, a ordem, da melhor para a pior rede, com base no somatório de suas detecções realizadas com PRDPs entre 1% e 3%, foi:

1. Pearson (42 falhas)
2. PCTN (34 falhas)
3. PCTN-MST (30 falhas)
4. MST (26 falhas)
5. MST-PCTN (6 falhas)

Falhas de desempenho

Nesta avaliação, simulamos uma falha de desempenho inserindo um atraso no carregamento das páginas listadas na Tabela 4.2. Esse atraso foi gerado através da inserção de um método que faz as *threads* responsáveis pela construção das páginas permanecerem em estado de dormência por 9 segundos antes de iniciarem o procedimento de carregamento.

Para essa avaliação, realizamos 14 experimentos e, em cada um deles, uma página diferente teve o atraso inserido. A Figura 4.12 mostra os resultados dos testes para valores de PRDP variando de 0 a 10%. Podemos observar que as redes de Pearson, PCTN e PCTN-MST obtiveram seus melhores resultados com $PRDP = 3\%$. Pearson detectou todas as falhas de desempenho e as redes PCTN e PCTN-MST detectaram

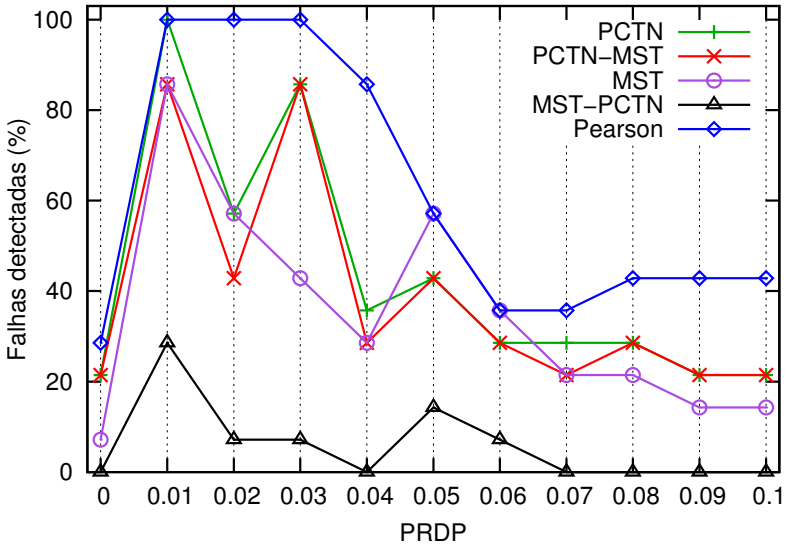


Figura 4.11: *Porcentagem de falhas detectadas por cada rede nos experimentos de falha de software.*

aproximadamente 85% e 78%, respectivamente. Para as redes MST e MST-PCTN, a melhor PRDP foi de 2%, na qual essas redes detectaram aproximadamente 65% e 30%, respectivamente. Dessa forma, nesta avaliação, a ordem, da melhor para a pior rede, com base no somatório de suas detecções realizadas com PRDPs entre 1% e 3%, foi:

1. Pearson (42 falhas)
2. PCTN (29 falhas)
3. PCTN-MST (27 falhas)
4. MST (22 falhas)
5. MST-PCTN (5 falhas)

Falha de configuração

Nesta avaliação, realizamos um único experimento no qual a quantidade de memória disponível para o Tomcat foi reduzida de 2 GB para 256 MB. A Figura 4.13 mostra como as redes de monitoramento reagiram à introdução dessa falha. Novamente, apresentamos os resultados para os valores de PRDP entre 0 e 10%. Podemos perceber que com $PRDP = 0\%$, todas as redes, com exceção da MST-PCTN, detectaram a falha de configuração. No entanto, para os valores de $PRDP >= 1\%$, somente a rede de Pearson foi capaz de detectar a falha. Dessa forma, nesta avaliação, a ordem, da melhor para a pior rede, com base no somatório de suas detecções realizadas com PRDPs entre 1% e 3%, foi:

1. Pearson (1 falha)

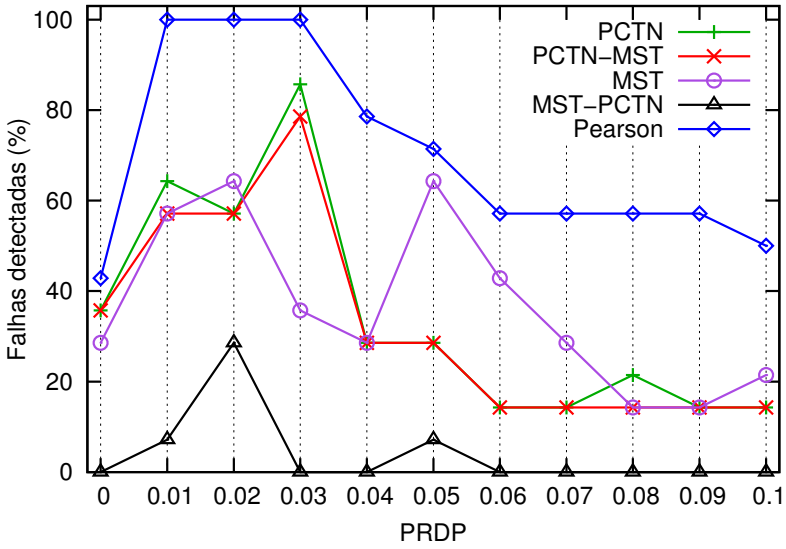


Figura 4.12: *Porcentagem de falhas detectadas por cada rede nos experimentos de falha de desempenho.*

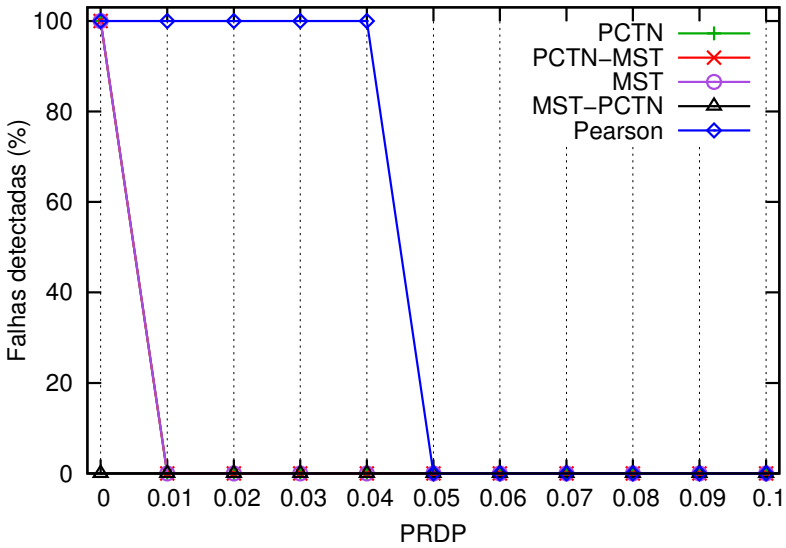


Figura 4.13: *Porcentagem de falhas detectadas por cada rede nos experimentos de falha de configuração.*

2. PCTN, PCTN-MST, MST, MST-PCTN (0 falhas)

Os resultados sugerem que apenas uma única correlação, capaz de detectar essa falha, se mostrou estável e, portanto, permaneceu nas redes de monitoramento PCTN, PCTN-MST, MST e Pearson, a saber, a correlação entre as métricas “front-GenericJMX-memory-memory-heap-used.value” e ”front-GenericJMX-memory_pool-CMS Old Gen-memory-used.value”. No entanto os resultados também mostraram que a perturbação causada nessa correlação foi muito pequena, o que justifica o fato dessa correlação só ter sido capaz de detectar a falha com $PRDP = 0\%$. Pearson conseguiu detectar a falha com $PRDP > 0\%$ devido a outras correlações não capturadas pelas outras redes.

Falhas de comunicação

Nesta avaliação, realizamos dois experimentos que simulam falhas de comunicação na rede que conecta o *Front Server* e o *Database Server*. No primeiro experimento, simulamos perdas de pacotes com a ferramenta *netem*. As perdas ocorrem com probabilidade de $0,01\%$, visto que, valores maiores que essa taxa podem afetar severamente o desempenho do *Transmission Control Protocol* (TCP), como pode ser visto no estudo de Augusto et al. [1]. No segundo experimento, simulamos atrasos de 90 a 110 milissegundos na entrega dos pacotes. A Figura 4.14 mostra o resultado dos dois experimentos para valores de PRDP entre 0 e 10%. Com algumas PRDPs entre 1% e 4%, as redes de Pearson, PCTN, PCTN-MST e MST detectaram as duas falhas de comunicação. A rede MST-PCTN, mais uma vez, apresentou o pior desempenho detectando apenas uma das duas falhas de comunicação. Dessa forma, nesta avaliação, a ordem da melhor para a pior rede, com base no somatório de suas detecções realizadas com PRDPs entre 1% e 3%, foi:

1. Pearson (6 falhas)
2. PCTN, PCTN-MST, MST (5 falhas)
3. MST-PCTN (3 falhas)

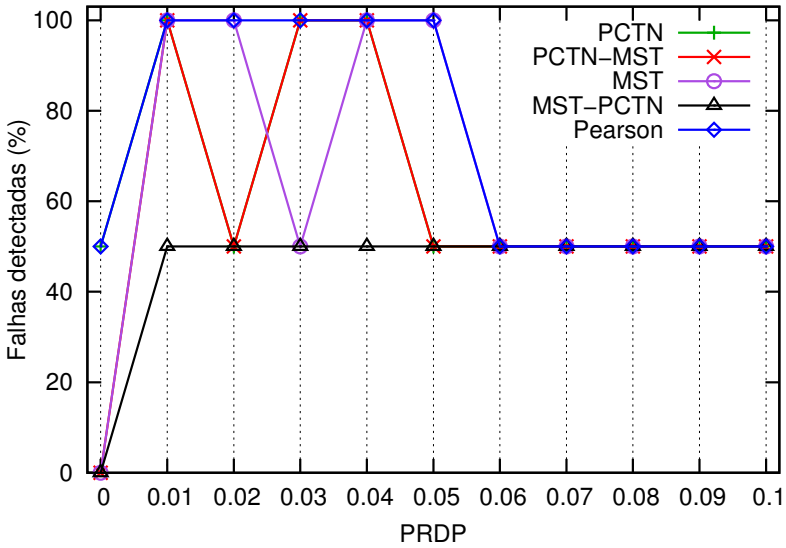


Figura 4.14: *Porcentagem de falhas detectadas por cada rede nos experimentos de falha de comunicação.*

Visão global dos resultados dos experimentos de detecção de falhas

Todas as redes de monitoramento obtiveram seus melhores resultados dentro do intervalo de PRDPs entre 1% e 3%. Portanto, nesta avaliação global que reuni todas as

falhas, nos atemos a esse intervalo. A Figura 4.15 apresenta as porcentagens de falhas detectadas, considerando todas as avaliações de falhas, para cada rede de monitoramento no intervalo de PRDPs de 1% a 3%. Como podemos observar, dentre as 31 falhas inseridas no sistema, a rede de Pearson foi capaz de detectar 100% delas para os três valores de PRDP. No entanto, a rede de Pearson possui quase 3 vezes mais correlações que a PCTN e 6 vezes mais correlações que a MST e a PCTN-MST.

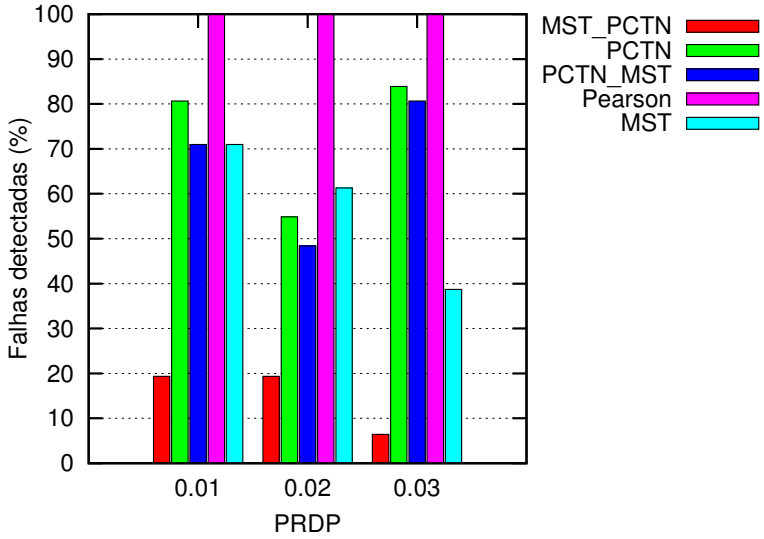


Figura 4.15: *Visão global das porcentagem de falhas detectadas por cada rede nas PRDPs com maiores níveis de detecção.*

Nossas propostas com melhor desempenho foram PCTN e PCTN-MST, as quais alcançaram a maior acurácia com $PRDP = 3\%$. Com essa parametrização, as redes PCTN e PCTN-MST conseguiram detectar mais de 80% das falhas. O melhor resultado da rede MST ocorreu com $PRDP = 1\%$, no qual a rede conseguiu detectar aproximadamente 70% das falhas. Portanto, duas das nossas propostas superam em 10% ou mais a MST. Além disso, a rede PCTN-MST possui aproximadamente 5% menos correlações e em torno de 8% menos métricas que a MST. Novamente, a rede MST-PCTN apresentou o pior desempenho e não conseguiu detectar mais que 20% das falhas em nenhum dos PRDPs avaliados. Esse resultado se deve ao número excessivamente pequeno de correlações e métricas mantidas por essa rede.

4.3.3 Avaliação de falso-positivos

Nesta avaliação, verificamos a porcentagem de falso-positivos emitidos pelas redes de monitoramento. Um falso-positivo é emitido quando uma rede sinaliza a ocorrência de falha em um período em que o sistema encontra-se em estado normal de operação.

Para essa avaliação executamos 31 experimentos livres de falhas. Cada experimento tem duração de 30 minutos. A Figura 4.16 mostra o resultado dos experimentos para os mesmos valores de PRDPs usados na avaliação de visão global de detecção de falhas, isto é, 1%, 2% e 3%. Podemos observar que a rede de Pearson, apesar de obter um ótimo desempenho na avaliação de detecção de falhas, também foi a rede que apresentou a maior porcentagem de falso-positivos. A MST-PCTN, que teve a pior acurácia na detecção de falhas, apresentou o melhor desempenho na taxa de falso-positivos.

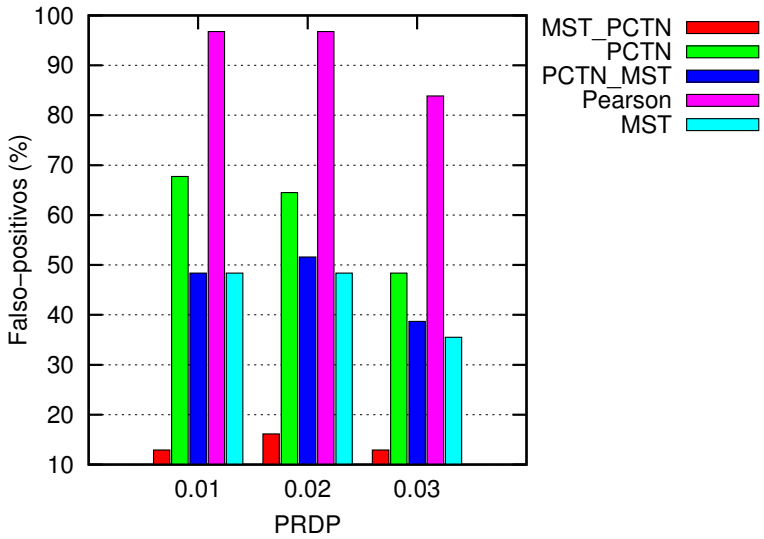


Figura 4.16: *Falso-positivos emitidos pelas redes de monitoramento.*

Em sua melhor configuração, $PRDP = 3\%$, a rede MST apresentou aproximadamente 35% de falso-positivos. Esse resultado supera por pouco o desempenho da rede PCTN-MST que atingiu 38%. No entanto, com essa mesma configuração, a PCTN-MST supera a MST em mais de 100% na taxa de detecção de falhas, como pode ser observado na Figura 4.15. A rede PCTN apresentou uma taxa de falso-positivo 37% maior que a da MST com $PRDP = 3\%$. Em contrapartida, a taxa de detecção de falhas da PCTN, com essa mesma configuração, chegou a aproximadamente 120% a mais de detecções que a MST. Dessa forma, nesta avaliação, a ordem, da melhor para a pior rede, com base no somatório de seus falso-positivos com PRDPs entre 1% e 3%, foi:

1. MST-PCTN (13 falso-positivos)
2. MST (41 falso-positivos)
3. PCTN-MST (43 falso-positivos)
4. PCTN (56 falso-positivos)
5. Pearson (86 falso-positivos)

É possível perceber que todas as redes de monitoramento obtiveram um alto índice de falso-positivo. Temos fortes indícios de que isso ocorre devido à breve duração

dos experimentos, os quais executam por apenas 30 minutos. Essa pequena duração leva à geração de muitos modelos não estáveis. Como podemos observar na Figura 4.9, quando os experimentos duram um período mais longo de tempo, por exemplo 360 minutos, há uma quantidade maior de modelos estáveis. Nesse contexto, todas as redes de monitoramento seriam beneficiadas com experimentos de tempos maiores.

4.4 Análise da eficiência das redes

Nesta seção, é apresentada a F-measure [11] das redes de monitoramento avaliadas. A F-measure tem como objetivo avaliar a eficiência das redes com base em seus verdadeiro-positivos, falso-positivos e falso-negativos. Ela também pode ser expressa por meio da *precision* (Equação 4-1) e da *recall* (Equação 4-2). Enquanto a *precision* determina a porcentagem do total de alarmes disparados que, de fato, envolveram falhas, a *recall* indica a porcentagem de falhas que foram detectadas.

$$precision = \frac{verdadeiro\_positivos}{verdadeiro\_positivos + falso\_positivos}. \tag{4-1}$$

$$recall = \frac{verdadeiro\_positivos}{verdadeiro\_positivos + falso\_negativos}. \tag{4-2}$$

A F-measure expressa através da *precision* e da *recall* é dada pela Equação 4-3:

$$F_\beta = (1+\beta^2) * \frac{P*R}{(\beta^2*P)+R}, \tag{4-3}$$

onde P e R são, respectivamente, a *precision* e a *recall*, e β é um parâmetro que controla o balanceamento entre P e R, podendo assumir os valores $0 \leq \beta \leq \infty$. Quando $\beta > 1$, há um majoração do peso do termo *recall* sobre o termo *precision*, ou seja, a F-measure torna-se mais orientada à *recall*. Em contrapartida, quando $\beta < 1$, a F-measure se torna mais orientada à *precision*. Quando $\beta = 1$, a F-measure se torna uma média harmônica de P e R. Além disso, a F-mesure fornece pontuações dentro do intervalo $[0,1]$, onde 0 representa o pior valor e 1 representa o melhor valor. Em nossa análise, nós utilizamos $\beta = 1$.

A Figura 4.17 apresenta os valores da F-measure para as PRDPs 1%, 2% e 3%. Como pode ser observado, a rede MST-PCTN foi a que obteve o menor valor F-measure em todas as PRDPs, fato relacionado ao baixo índice de detecção de falhas. Em contrapartida, a rede de Pearson, mesmo tendo um alto índice de falso-positivos, superou todas as redes nas PDRPs de 1% e 2%, e ficou em terceira colocação na PRDP 3%, sendo superada apenas pelas redes PCTN e PCTN-MST nessa parametrização. Esse bom resultado obtido pela rede de Pearson está relacionado ao seu bom desempenho na

detecção de falhas, que chegou a 100%. De qualquer forma, é importante notar que, dentre todas as redes em todas as parametrizações, as redes PCTN e PCTN-MST, com PRDP de 3%, foram as que obtiveram os maiores valores F-measure.

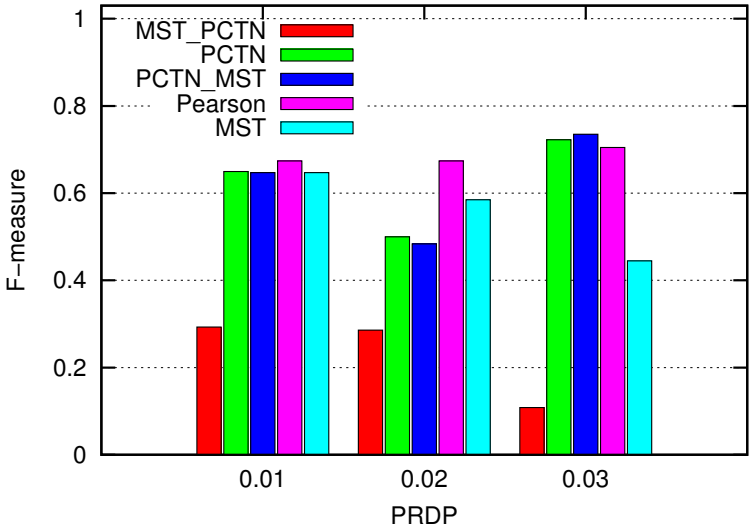


Figura 4.17: *F-measure das redes de monitoramento para as PRDPs* 1%*,* 2% *e* 3%*.*

4.5 Avaliação de tempo de resposta

Nesta avaliação, analisamos o tempo necessário para processar 30 minutos de monitoramento e responder se há ou não uma falha no sistema. A Figura 4.18 exibe o tempo médio gasto pelas redes para as PRDPs 1%, 2% e 3%. Como já esperávamos, as redes com maior número de correlações levaram mais tempo para serem processadas. Por meio do gráfico é possível visualizar que a rede de Pearson levou cerca de três vezes o tempo de processamento da PCTN e cerca de cinco vezes o tempo de processamento das redes MST e PCTN-MST para ser processada no pior cenário apresentado ($PRDP = 3\%$). A rede PCTN levou cerca de duas vezes o tempo de processamento das redes MST e PCTN-MST. Os tempos de processamento das redes MST e PCTN-MST permaneceram muito próximos, com uma leve vantagem, na média, para a rede PCTN-MST.

É importante notar que medir o tempo de processamento exato de cada uma das redes é uma tarefa complicada. Mesmo uma sutil diferença de temperatura ambiente pode alterar o *clock* da máquina e, consequentemente, inserir pequenas inconsistências nessa avaliação. Esse fato pode explicar, por exemplo, o motivo da rede MST-PCTN ter levado mais tempo para ser processada na parametrização $PRDP = 2\%$ do que na parametrização $PRDP = 3\%$, apesar da rede MST-PCTN com $PRDP = 3\%$ possuir todas as correlações existentes na rede MST-PCTN com a $PRDP = 2\%$ além de outras novas. Dessa forma,

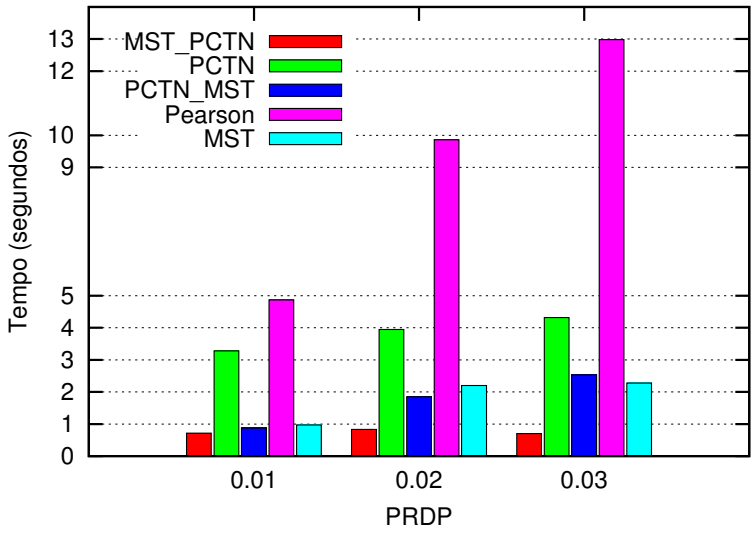


Figura 4.18: *Tempo de processamento despendido em cada rede para analisar* 360 *coletas estatísticas.*

os resultados desta seção devem ser vistos apenas como uma aproximação do tempo de processamento de cada uma das redes.

A seguir, as redes foram ordenadas de forma crescente pelo somatório dos tempos despendidos no monitoramento com base nas PRDPs com valores entre 1% e 3%:

1. MST-PCTN (2,25 segundos)
2. PCTN-MST (5,25 segundos)
3. MST (5,44 segundos)
4. PCTN (11,54 segundos)
5. Pearson (27,72 segundos)

4.6 Considerações finais

Neste capítulo, apresentamos nossa infraestrutura de testes, a metodologia de avaliação e a construção das redes de monitoramento. Além disso, também apresentamos as avaliações de estabilidade de modelos, detecção de falhas, falso-positivos e tempo de resposta. A partir dos resultados, pudemos concluir que duas das três redes baseadas em correlação parcial, em especial a rede PCTN-MST, obtiveram resultados satisfatórios em todas as avaliações. Além disso, como o custo de processamento no monitoramento cresce exponencialmente com o número de métricas [50], é de suma importância a utilização de métodos para a filtragem de correlações pertencentes às redes de monitoramento. No entanto isso deve ser feito sem que haja impacto significativo na eficiência da rede. Nesse sentido, uma de nossas propostas, a saber, a rede de monitoramento PCTN-MST

conseguiu obter uma menor quantidade de correlações e métricas que as redes Pearson e MST e, ainda assim, manter uma taxa de detecção de falhas e um índice de falso-positivos satisfatórios. Por esse motivo, com base nas avaliações de detecção de falhas, falso-positivos e tempo de resposta, nós consideramos que a rede PCTN-MST com PRDP 3% foi a rede com o melhor custo-benefício.

CAPÍTULO 5

Conclusão e Trabalhos Futuros

Neste trabalho, apresentamos três filtros de métricas monitoradas por mecanismos de detecção de falhas que atuam sobre aplicações *Web*. Nossas propostas usam o conceito de correlação parcial para identificar um conjunto reduzido de métricas que permitem identificar falhas, visando um menor consumo de processamento no monitoramento. Apesar da correlação parcial já ter sido utilizada para extrair informações de outros ambientes complexos de outras áreas, como no estudo de genes [8, 15, 39], em redes de neurônios [18] e, mais recentemente, em redes de mercados financeiros [25, 43], até onde sabemos, este é o primeiro trabalho que utiliza a correlação parcial em um ambiente computacional.

Para implementar os filtros de métricas e correlações baseados em correlação parcial, propomos três estratégias. A primeira, denominada PCTN, constrói uma rede de correlação baseada nas influências de correlação. A segunda estratégia, denominada MST-PCTN, constrói uma rede de correlações baseada na MST e nas influências de correlações, onde as métricas selecionadas na rede MST são filtradas, posteriormente, pela PCTN. A terceira estratégia, denominada PCTN-MST, constrói uma rede de correlação baseada nas influências de correlações e na MST, onde as correlações selecionadas pela rede PCTN são filtradas, posteriormente, pela MST.

Para avaliar os três filtros, implantamos uma *testbed* que simula um ambiente *Web* em três camadas. Nesse ambiente, o ambiente *Web* e seus clientes foram simulados através do *benchmark* TPC-W. Em seguida, comparamos o desempenho dos filtros propostos (PCTN, MST-PCTN e PCTN-MST) com soluções da literatura, a saber: os filtros baseados em Pearson e na MST.

A avaliação experimental das cinco redes de monitoramento, geradas a partir dos cinco filtros implementados, foi realizada considerando os seguintes aspectos: estabilidade dos modelos, capacidade de detecção de anomalias, emissão de falso-positivos e tempo médio para as redes emitirem uma resposta.

Os resultados mostram que Pearson foi a solução que conseguiu detectar o maior número de falhas, porém, foi também a rede com o maior índice de falso-positivos. Uma de nossas propostas, a rede MST-PCTN, apesar de ter atingido índices de falso-positivos

baixíssimos, não conseguiu detectar nem mesmo 20% das falhas. As soluções PCTN e PCTN-MST conseguiram ampliar em mais de 10% o índice de detecção de falhas da rede MST, mantendo, praticamente, o mesmo índice de falso-positivos. Além disso, a rede PCTN-MST resultou em 8% menos métricas e em 5% menos correlações que a rede MST. A rede PCTN atingiu um índice de detecção de falhas quase 5% maior que a PCTN-MST ao custo de ampliar em aproximadamente 10% o índice de falso-positivos. Vale ressaltar que, apesar da redução no número de correlações e métricas ter sido relativamente pequena em nossa avaliação, nosso ambiente de testes é composto de apenas dois servidores e, como o número de correlações tende a crescer de forma exponencial com o número de métricas [50], em ambientes maiores, a redução também tende a ser mais impactante.

Por fim, apesar da avaliação de nossos filtros ter ficado restrita a um ambiente *Web*, não existe nenhum empecilho de aplicá-los a outros ambientes computacionais.

5.1 Perspectivas para trabalhos futuros

Como mencionado no Capítulo 4, nossos filtros de métricas são projetados para trabalhar com mecanismos de detecção de anomalias que utilizam monitoramento baseado em correlação. Nesses mecanismos, as correlações são derivadas a partir de procedimentos realizados de forma *offline*. Em ambientes onde a carga de trabalho é dinâmica, mecanismos de monitoramento baseados em correlação podem ter dificuldade em capturar a dinâmica do ambiente de execução. Portanto, uma perspectiva de evolução deste trabalho é a investigação de maneiras de detectar mudanças na carga de trabalho e a atualização *online* dos modelos para refletir essas novas condições.

Outro desdobramento dessa pesquisa diz respeito ao diagnóstico de falhas. Neste trabalho, limitamos o escopo da nossa abordagem à fase de detecção de anomalias. Uma etapa importante e complementar envolve a aplicação de técnicas que permitam rastrear as métricas que violam uma invariante a fim de diagnosticar a falha detectada.

Referências Bibliográficas

[1] AUGUSTO, C. H. P.; DA SILVA, M. W. R.; CARDOSO, K. V.; MENDES, A. C.; GUEDES, R. M.; DE REZENDE, J. F. **Segmentação de conexões tcp para a transferência fim-a-fim em alta velocidade**. In: *Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance), XXVIII Congresso da Sociedade Brasileira de Computação (CSBC)*, 2008.

[2] BABA, K.; SHIBATA, R.; SIBUYA, M. **Partial correlation and conditional correlation as measures of conditional independence**. *Australian & New Zealand Journal of Statistics*, 46:657–664, 2004.

[3] 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 & Implementation - Volume 6*, OSDI'04, p. 18–18, Berkeley, CA, USA, 2004. USENIX Association.

[4] BEZENEK, T.; CAIN, T.; DICKSON, R.; HEIL, T.; MARTIN, M.; MCCURDY, C.; RAJWAR, R.; WEGLARZ, E.; ZILLES, C.; LIPASTI, M. **Java TPC-W Implementation Distribution**. `http://pharm.ece.wisc.edu/tpcw.shtml`, June 2012. [Último acesso: 21-Setembro-2014].

[5] CHANDOLA, V.; BANERJEE, A.; KUMAR, V. **Anomaly detection: A survey**. *ACM Comput. Surv.*, 41(3):15:1–15:58, jul 2009.

[6] CHEN, H.; JIANG, G.; UNGUREANU, C.; YOSHIHIRA, K. **Failure detection and localization in component based systems by online tracking**. In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, p. 750–755, New York, NY, USA, 2005. ACM.

[7] CHEN, H.; JIANG, G.; YOSHIHIRA, K.; SAXENA, A. **Invariants based failure diagnosis in distributed computing systems**. In: *Reliable Distributed Systems, 2010 29th IEEE Symposium on*, p. 160–166, 2010.

[8] CHEN, L.; ZHENG, S. **Studying alternative splicing regulatory networks through partial correlation analysis**. *Genome Biology*, 10(1):R3, 2009.

[9] CHEN, M. Y.; ACCARDI, A.; KICIMAN, E.; LLOYD, J.; PATTERSON, D.; FOX, A.; BREWER, E. **Path-based faliure and evolution management**. In: *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, p. 23–23, Berkeley, CA, USA, 2004. USENIX Association.

[10] CHERKASOVA, L.; OZONAT, K.; MI, N.; SYMONS, J.; SMIRNI, E. **Automated anomaly detection and performance modeling of enterprise applications**. *ACM Transactions on Computer Systems, (TOCS)*, 27(3):6:1–6:32, november 2009.

[11] CHINCHOR, N. **Muc-4 evaluation metrics**. In: *Proceedings of the 4th Conference on Message Understanding*, MUC4 '92, p. 22–29. Association for Computational Linguistics, 1992.

[12] COHEN, I.; GOLDSZMIDT, M.; KELLY, T.; SYMONS, J.; CHASE, J. S. **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 & Implementation - Volume 6*, OSDI'04, p. 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[13] COHEN, J. **Statistical Power Analysis for the Behavioral Sciences**. Lawrence Erlbaum Associates, 2nd edition edition, jan 1988.

[14] COLLECTD. **collectd**. `http://collectd.org`, 2014. [Último acesso: 04-Maio-2014].

[15] DE LA FUENTE, A.; BING, N.; HOESCHELE, I.; MENDES, P. **Discovery of meaningful associations in genomic data using partial correlation coefficients**. *Bioinformatics*, 20(18):3565–3574, Dec. 2004.

[16] DOOLEY, R. **Don't Let a Slow Website Kill Your Bottom Line**. `http://www.forbes.com/sites/rogerdooley/2012/12/04/fast-sites/`, Apr. 2012. [Último acesso: 14-Maio-2014].

[17] FEITELSON, D. G. **Workload Modeling for Computer Systems Performance Evaluation**. 1.0.1 edition, 2014.

[18] FRANSSON, P.; MARRELEC, G. **The precuneus/posterior cingulate cortex plays a pivotal role in the default mode network: Evidence from a partial correlation network analysis**. *NeuroImage*, 42(3):1178–1184, Sept. 2008.

[19] GHANBARI, S.; AMZA, C. **Semantic-driven model composition for accurate anomaly diagnosis**. In: *Autonomic Computing, 2008. ICAC '08. International Conference on*, p. 35–44, 2008.

[20] GUO, Z.; JIANG, G.; CHEN, H.; YOSHIHIRA, K. **Tracking probabilistic correlation of monitoring data for fault detection in complex systems**. In: *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, p. 259–268, 2006.

[21] IBM. **IBM SmartCloud Analytics**. http://www-03.ibm.com/software/products/en/ibm-smartcloud-analytics---predictive-insights, 2014. [Último acesso: 15-Junho-2014].

[22] JIANG, G.; CHEN, H.; YOSHIHIRA, K. **Discovering likely invariants of distributed transaction systems for autonomic system management**. *Cluster Computing*, 9(4):385–399, Oct. 2006.

[23] JIANG, G.; CHEN, H.; YOSHIHIRA, K. **Modeling and tracking of transaction flow dynamics for fault detection in complex systems**. *IEEE Trans. Dependable Secur. Comput.*, 3(4):312–326, Oct. 2006.

[24] JIANG, G.; CHEN, H.; YOSHIHIRA, K.; SAXENA, A. **Ranking the importance of alerts for problem determination in large computer systems**. *Cluster Computing*, 14(3):213–227, 2011.

[25] KENETT, D. Y.; TUMMINELLO, M.; MADI, A.; GUR-GERSHGOREN, G.; MANTEGNA, R. N.; BEN-JACOB, E. **Dominating clasp of the financial sector revealed by partial correlation analysis of the stock market**. *PLoS ONE*, 5(12):e15032, 2010.

[26] LAING, B. **Summary of Windows Azure Service Disruption on Feb 29th, 2012**. http://azure.microsoft.com/blog/2012/03/09/summary-of-windows-azure-service-disruption-on-feb-29th-2012/, Feb. 2012. [Último acesso: 12-Junho-2014].

[27] LINUX FOUNDATION. **netem**. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem, Nov. 2009. [Último acesso: 20-Setembro-2014].

[28] MAGALHAES, J.; MOURA SILVA, L. **Adaptive profiling for root-cause analysis of performance anomalies in web-based applications**. In: *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, p. 171–178, 2011.

[29] MENASCE, D. **Tpc-w: a benchmark for e-commerce**. *Internet Computing, IEEE*, 6(3):83–87, 2002.

[30] MICROSOFT. **System Center Operations Manager**. http://technet.microsoft.com/en-us/systemcenter/bb497976, 2013. [Último acesso: 16-Agosto-2013].

[31] MILLER, R. **Performance Issues for Amazon.com**. http://www.datacenterknowledge.com/archives/2010/06/29/performance-issues-for-amazon-com/, June 2010. [Último acesso: 12-Junho-2014].

[32] MILLER, R. **The Facebook DataCenter FAQ**. http://www.datacenterknowledge.com/the-facebook-data-center-faq/, Sept. 2010. [Último acesso: 19-Agosto-2014].

[33] MILLER, R. **A Look Inside Amazon¿s Data Centers**. http://www.datacenterknowledge.com/archives/2011/06/09/a-look-inside-amazons-data-centers/, June 2011. [Último acesso: 19-Agosto-2014].

[34] MILLER, R. **Google Data Center FAQ**. http://www.datacenterknowledge.com/archives/2012/05/15/google-data-center-faq/, May 2012. [Último acesso: 19-Agosto-2014].

[35] MONTGOMERY, D.; PECK, E.; VINING, G. **Introduction to Linear Regression Analysis**. Wiley Series in Probability and Statistics. Wiley, 2012.

[36] MUNAWAR, M. A.; JIANG, M.; REIDEMEISTER, T.; WARD, P. A. **Filtering system metrics for minimal correlation-based self-monitoring**. In: *Self-Adaptive and Self-Organizing Systems, 2009. SASO '09. Third IEEE International Conference on*, p. 233–242, 2009.

[37] PERTET, S.; NARASIMHAN, P. **Causes of failure in web applications**. Technical Report CMU-PDL-05-109, Parallel Data Laboratory, Carnegie Mellon University, Dec. 2005.

[38] PINNO, O. J. A.; CORRÊA, S. L.; DO SACRAMENTO RODRIGUES, V. J.; CARDOSO, K. V. **Abordagem Baseada em Correlação Parcial para Filtragem de Métricas Monitoradas em Sistemas Auto-Gerenciáveis**. In: *Anais do 4º Workshop de Sistemas Distribuídos Autonômicos - WoSiDA 2014*, p. 39–42, May 2014.

[39] REVERTER, A.; CHAN, E. K. F. **Combining partial correlation and an information theory approach to the reversed engineering of gene co-expression networks**. *Bioinformatics*, 24(21):2491–2497, 2008.

[40] SEDGEWICK, R.; WAYNE, K. **Introduction to programming in java**. http://introcs.cs.princeton.edu/java/home/, Apr. 2013. [Último acesso: 25-Setembro-2014].

[41] SERVICES, A. W. **Amazon EC2**. http://aws.amazon.com/ec2/, Nov. 2014. [Último acesso: 04-Novembro-2014].

[42] SHANKLAND, S. **PayPal suffers from e-commerce outage**. http://www.cnet.com/news/paypal-suffers-from-e-commerce-outage/, Aug. 2009. [Último acesso: 16-Junho-2014].

[43] SHAPIRA, Y.; KENETT, D. Y.; BEN-JACOB, E. **The index cohesive effect on stock market correlations**. *The European Physical Journal B*, 72(4):657–669, 2009.

[44] SHATNAWI, M.; RIPEANU, M. **Failure avoidance through fault prediction based on synthetic transactions**. In: *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2011, Newport Beach, CA, USA, May 23-26, 2011*, p. 324–331. IEEE, 2011.

[45] TAN, Y.; NGUYEN, H.; SHEN, Z.; GU, X.; VENKATRAMANI, C.; RAJAN, D. **Prepare: Predictive performance anomaly prevention for virtualized cloud systems**. In: *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, p. 285–294, June 2012.

[46] URGAONKAR, B.; PACIFICI, G.; SHENOY, P.; SPREITZER, M.; TANTAWI, A. **An analytical model for multi-tier internet services and its applications**. *SIGMETRICS Perform. Eval. Rev.*, 33(1):291–302, June 2005.

[47] WANG, C.; KAVULYA, S. P.; TAN, J.; HU, L.; KUTARE, M.; KASICK, M.; SCHWAN, K.; NARASIMHAN, P.; GANDHI, R. **Performance troubleshooting in data centers: An annotated bibliography?** *SIGOPS Oper. Syst. Rev.*, 47(3):50–62, Nov. 2013.

[48] WANG, T.; WEI, J.; ZHANG, W.; ZHONG, H.; HUANG, T. **Workload-aware anomaly detection for web applications**. *The Journal of Systems and Software*, 89:19–32, Mar. 2014.

[49] WEINREICH, H.; OBENDORF, H.; HERDER, E.; MAYER, M. **Not quite the average: An empirical study of web use**. *ACM Trans. Web*, 2(1):5:1–5:31, Mar. 2008.

[50] WITTEN, I. H.; FRANK, E. **Data Mining: Practical Machine Learning Tools and Techniques**. Morgan Kaufmann, second edition edition, June 2005.

[51] YUAN, D.; ZHENG, J.; PARK, S.; ZHOU, Y.; SAVAGE, S. **Improving software diagnosability via log enhancement**. *SIGARCH Comput. Archit. News*, 39(1):3–14, Mar. 2011.

[52] ZHANG, Q.; CHERKASOVA, L.; MATHEWS, G.; GREENE, W.; SMIRNI, E. **R-capriccio: a capacity planning and anomaly detection tool for enterprise services with live workloads**. In: *Proceedings of the ACM/IFIP/USENIX 2007 International Conference*

on Middleware, Middleware '07, p. 244–265, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

[53] ZHANG, Q.; CHERKASOVA, L.; SMIRNI, E. **A regression-based analytic model for dynamic resource provisioning of multi-tier applications**. In: *Proceedings of the Fourth International Conference on Autonomic Computing*, ICAC '07, p. 27–, Washington, DC, USA, 2007. IEEE Computer Society.

[54] 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, New York, NY, USA, 2007. ACM.

[55] ZHANG, S.; COHEN, I.; GOLDSZMIDT, M.; SYMONS, J.; FOX, A. **Ensembles of models for automated diagnosis of system performance problems**. In: *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, p. 644–653, 2005.

APÊNDICE A

Métricas coletadas do ambiente de testes

Neste apêndice encontram-se todas as métricas que foram coletadas em nosso ambiente de testes. Essas métricas foram coletadas com o software collectd e foram utilizadas na validação dos filtros de métricas propostos por este trabalho.

Na tabela a seguir são especificados o equipamento, o arquivo e o item, que juntos compõem cada uma das métricas coletas. O equipamento "back" refere-se ao servidor de banco de dados, enquanto que o equipamento "front" refere-se ao servidor Web e de aplicação.

Equipamento	***Arquivo***	***Item***
back	cpu-0-average_usage	value
back	cpu-0-cpu-idle	value
back	cpu-0-cpu-interrupt	value
back	cpu-0-cpu-nice	value
back	cpu-0-cpu-softirq	value
back	cpu-0-cpu-steal	value
back	cpu-0-cpu-system	value
back	cpu-0-cpu-user	value
back	cpu-0-cpu-wait	value
back	cpu-1-average_usage	value
back	cpu-1-cpu-idle	value
back	cpu-1-cpu-interrupt	value
back	cpu-1-cpu-nice	value
back	cpu-1-cpu-softirq	value
back	cpu-1-cpu-steal	value
back	cpu-1-cpu-system	value
back	cpu-1-cpu-user	value
back	cpu-1-cpu-wait	value
back	cpu-2-average_usage	value
back	cpu-2-cpu-idle	value

back	cpu-2-cpu-interrupt	value
back	cpu-2-cpu-nice	value
back	cpu-2-cpu-softirq	value
back	cpu-2-cpu-steal	value
back	cpu-2-cpu-system	value
back	cpu-2-cpu-user	value
back	cpu-2-cpu-wait	value
back	cpu-3-average_usage	value
back	cpu-3-cpu-idle	value
back	cpu-3-cpu-interrupt	value
back	cpu-3-cpu-nice	value
back	cpu-3-cpu-softirq	value
back	cpu-3-cpu-steal	value
back	cpu-3-cpu-system	value
back	cpu-3-cpu-user	value
back	cpu-3-cpu-wait	value
back	cpu-average_usage	value
back	cpufreq-average	value
back	cpufreq-cpufreq-0	value
back	cpufreq-cpufreq-1	value
back	cpufreq-cpufreq-2	value
back	cpufreq-cpufreq-3	value
back	disk-sda-disk_merged	read
back	disk-sda-disk_merged	write
back	disk-sda-disk_octets	read
back	disk-sda-disk_octets	write
back	disk-sda-disk_ops	read
back	disk-sda-disk_ops	write
back	disk-sda-disk_time	read
back	disk-sda-disk_time	write
back	entropy-entropy	entropy
back	interface-if_errors-eth1	rx
back	interface-if_errors-eth1	tx
back	interface-if_octets-eth1	rx
back	interface-if_octets-eth1	tx
back	interface-if_packets-eth1	rx

back	interface-if_packets-eth1	tx
back	memory-memory-buffered	value
back	memory-memory-cached	value
back	memory-memory-free	value
back	memory-memory-used	value
back	mysql-tpcw-mysql_commands-admin_commands	value
back	mysql-tpcw-mysql_commands-delete	value
back	mysql-tpcw-mysql_commands-insert	value
back	mysql-tpcw-mysql_commands-select	value
back	mysql-tpcw-mysql_commands-set_option	value
back	mysql-tpcw-mysql_commands-show_collations	value
back	mysql-tpcw-mysql_commands-show_status	value
back	mysql-tpcw-mysql_commands-show_variables	value
back	mysql-tpcw-mysql_commands-update	value
back	mysql-tpcw-mysql_handler-commit	value
back	mysql-tpcw-mysql_handler-delete	value
back	mysql-tpcw-mysql_handler-read_first	value
back	mysql-tpcw-mysql_handler-read_key	value
back	mysql-tpcw-mysql_handler-read_last	value
back	mysql-tpcw-mysql_handler-read_next	value
back	mysql-tpcw-mysql_handler-read_rnd	value
back	mysql-tpcw-mysql_handler-read_rnd_next	value
back	mysql-tpcw-mysql_handler-update	value
back	mysql-tpcw-mysql_handler-write	value
back	mysql-tpcw-mysql_locks-immediate	value
back	mysql-tpcw-mysql_locks-waited	value
back	mysql-tpcw-mysql_octets	rx
back	mysql-tpcw-mysql_octets	tx
back	mysql-tpcw-mysql_qcache	hits
back	mysql-tpcw-mysql_qcache	inserts
back	mysql-tpcw-mysql_qcache	not_cached
back	mysql-tpcw-mysql_qcache	lowmem_prunes
back	mysql-tpcw-mysql_qcache	queries_in_cache
back	mysql-tpcw-mysql_threads	running
back	mysql-tpcw-mysql_threads	connected
back	mysql-tpcw-mysql_threads	cached

back	mysql-tpcw-mysql_threads	created
back	ping-ping-192.168.11.1	ping
back	ping-ping_droprate-192.168.11.1	value
back	ping-ping_stddev-192.168.11.1	value
back	processes-fork_rate	value
back	processes-mysqld-ps_code	value
back	processes-mysqld-ps_count	processes
back	processes-mysqld-ps_count	threads
back	processes-mysqld-ps_cputime	user
back	processes-mysqld-ps_cputime	syst
back	processes-mysqld-ps_data	value
back	processes-mysqld-ps_disk_octets	read
back	processes-mysqld-ps_disk_octets	write
back	processes-mysqld-ps_disk_ops	read
back	processes-mysqld-ps_disk_ops	write
back	processes-mysqld-ps_pagefaults	minflt
back	processes-mysqld-ps_pagefaults	majflt
back	processes-mysqld-ps_rss	value
back	processes-mysqld-ps_stacksize	value
back	processes-mysqld-ps_vm	value
back	processes-ps_state-blocked	value
back	processes-ps_state-paging	value
back	processes-ps_state-running	value
back	processes-ps_state-sleeping	value
back	processes-ps_state-stopped	value
back	processes-ps_state-zombies	value
back	tcpconns-3306-local-tcp_connections-CLOSED	value
back	tcpconns-3306-local-tcp_connections-CLOSE_WAIT	value
back	tcpconns-3306-local-tcp_connections-CLOSING	value
back	tcpconns-3306-local-tcp_connections-ESTABLISHED	value
back	tcpconns-3306-local-tcp_connections-FIN_WAIT1	value
back	tcpconns-3306-local-tcp_connections-FIN_WAIT2	value
back	tcpconns-3306-local-tcp_connections-LAST_ACK	value
back	tcpconns-3306-local-tcp_connections-LISTEN	value
back	tcpconns-3306-local-tcp_connections-SYN_RECV	value
back	tcpconns-3306-local-tcp_connections-SYN_SENT	value

back	tcpconns-3306-local-tcp_connections-TIME_WAIT	value
back	vmem-direct_dma-vmpage_action-steal	value
back	vmem-direct_dma32-vmpage_action-steal	value
back	vmem-direct_movable-vmpage_action-steal	value
back	vmem-direct_normal-vmpage_action-steal	value
back	vmem-dma-vmpage_action-alloc	value
back	vmem-dma-vmpage_action-refill	value
back	vmem-dma-vmpage_action-scan_direct	value
back	vmem-dma-vmpage_action-scan_kswapd	value
back	vmem-dma32-vmpage_action-alloc	value
back	vmem-dma32-vmpage_action-refill	value
back	vmem-dma32-vmpage_action-scan_direct	value
back	vmem-dma32-vmpage_action-scan_kswapd	value
back	vmem-kswapd_dma-vmpage_action-steal	value
back	vmem-kswapd_dma32-vmpage_action-steal	value
back	vmem-kswapd_movable-vmpage_action-steal	value
back	vmem-kswapd_normal-vmpage_action-steal	value
back	vmem-movable-vmpage_action-alloc	value
back	vmem-movable-vmpage_action-refill	value
back	vmem-movable-vmpage_action-scan_direct	value
back	vmem-movable-vmpage_action-scan_kswapd	value
back	vmem-normal-vmpage_action-alloc	value
back	vmem-normal-vmpage_action-refill	value
back	vmem-normal-vmpage_action-scan_direct	value
back	vmem-normal-vmpage_action-scan_kswapd	value
back	vmem-vmpage_action-activate	value
back	vmem-vmpage_action-deactivate	value
back	vmem-vmpage_action-free	value
back	vmem-vmpage_faults	minflt
back	vmem-vmpage_faults	majflt
back	vmem-vmpage_io-memory	in
back	vmem-vmpage_io-memory	out
back	vmem-vmpage_io-swap	in
back	vmem-vmpage_io-swap	out
back	vmem-vmpage_number-active_anon	value
back	vmem-vmpage_number-active_file	value

back	vmem-vmpage_number-anon_pages	value
back	vmem-vmpage_number-anon_transparent_hugepages	value
back	vmem-vmpage_number-bounce	value
back	vmem-vmpage_number-dirtied	value
back	vmem-vmpage_number-dirty	value
back	vmem-vmpage_number-dirty_background_threshold	value
back	vmem-vmpage_number-dirty_threshold	value
back	vmem-vmpage_number-file_pages	value
back	vmem-vmpage_number-free_pages	value
back	vmem-vmpage_number-inactive_anon	value
back	vmem-vmpage_number-inactive_file	value
back	vmem-vmpage_number-isolated_anon	value
back	vmem-vmpage_number-isolated_file	value
back	vmem-vmpage_number-kernel_stack	value
back	vmem-vmpage_number-mapped	value
back	vmem-vmpage_number-mlock	value
back	vmem-vmpage_number-page_table_pages	value
back	vmem-vmpage_number-shmem	value
back	vmem-vmpage_number-slab_reclaimable	value
back	vmem-vmpage_number-slab_unreclaimable	value
back	vmem-vmpage_number-unevictable	value
back	vmem-vmpage_number-unstable	value
back	vmem-vmpage_number-vmscan_immediate_reclaim	value
back	vmem-vmpage_number-vmscan_write	value
back	vmem-vmpage_number-writeback	value
back	vmem-vmpage_number-writeback_temp	value
back	vmem-vmpage_number-written	value
front	GenericJMX-gauge-loaded_classes	value
front	GenericJMX-gc-ConcurrentMarkSweep-invocations	value
front	GenericJMX-gc-ConcurrentMarkSweep-total_time_in_ms-collection_time	value
front	GenericJMX-gc-ParNew-invocations	value
front	GenericJMX-gc-ParNew-total_time_in_ms-collection_time	value
front	GenericJMX-memory-memory-heap-committed	value
front	GenericJMX-memory-memory-heap-init	value

front	GenericJMX-memory-memory-heap-max	value
front	GenericJMX-memory-memory-heap-used	value
front	GenericJMX-memory-memory-nonheap-committed	value
front	GenericJMX-memory-memory-nonheap-init	value
front	GenericJMX-memory-memory-nonheap-max	value
front	GenericJMX-memory-memory-nonheap-used	value
front	GenericJMX-memory_pool-CMS Old Gen-memory-committed	value
front	GenericJMX-memory_pool-CMS Old Gen-memory-init	value
front	GenericJMX-memory_pool-CMS Old Gen-memory-max	value
front	GenericJMX-memory_pool-CMS Old Gen-memory-used	value
front	GenericJMX-memory_pool-CMS Perm Gen-memory-committed	value
front	GenericJMX-memory_pool-CMS Perm Gen-memory-init	value
front	GenericJMX-memory_pool-CMS Perm Gen-memory-max	value
front	GenericJMX-memory_pool-CMS Perm Gen-memory-used	value
front	GenericJMX-memory_pool-Code Cache-memory-committed	value
front	GenericJMX-memory_pool-Code Cache-memory-init	value
front	GenericJMX-memory_pool-Code Cache-memory-max	value
front	GenericJMX-memory_pool-Code Cache-memory-used	value
front	GenericJMX-memory_pool-Par Eden Space-memory-committed	value
front	GenericJMX-memory_pool-Par Eden Space-memory-init	value
front	GenericJMX-memory_pool-Par Eden Space-memory-max	value
front	GenericJMX-memory_pool-Par Eden Space-memory-used	value
front	GenericJMX-memory_pool-Par Survivor Space-memory-committed	value
front	GenericJMX-memory_pool-Par Survivor Space-memory-init	value
front	GenericJMX-memory_pool-Par Survivor Space-memory-max	value
front	GenericJMX-memory_pool-Par Survivor Space-memory-used	value

front	GenericJMX-request_processor-"http-bio-8080-io_octets-global	rx
front	GenericJMX-request_processor-"http-bio-8080-io_octets-global	tx
front	GenericJMX-request_processor-"http-bio-8080-threads-running	value
front	GenericJMX-request_processor-"http-bio-8080-threads-total	value
front	GenericJMX-request_processor-"http-bio-8080-total_requests-global	value
front	GenericJMX-request_processor-"http-bio-8080-total_time_in_ms-global-processing	value
front	GenericJMX-total_time_in_ms-compilation_time	value
front	cpu-0-average_usage	value
front	cpu-0-cpu-idle	value
front	cpu-0-cpu-interrupt	value
front	cpu-0-cpu-nice	value
front	cpu-0-cpu-softirq	value
front	cpu-0-cpu-steal	value
front	cpu-0-cpu-system	value
front	cpu-0-cpu-user	value
front	cpu-0-cpu-wait	value
front	cpu-1-average_usage	value
front	cpu-1-cpu-idle	value
front	cpu-1-cpu-interrupt	value
front	cpu-1-cpu-nice	value
front	cpu-1-cpu-softirq	value
front	cpu-1-cpu-steal	value
front	cpu-1-cpu-system	value
front	cpu-1-cpu-user	value
front	cpu-1-cpu-wait	value
front	cpu-2-average_usage	value
front	cpu-2-cpu-idle	value
front	cpu-2-cpu-interrupt	value
front	cpu-2-cpu-nice	value
front	cpu-2-cpu-softirq	value

front	cpu-2-cpu-steal	value
front	cpu-2-cpu-system	value
front	cpu-2-cpu-user	value
front	cpu-2-cpu-wait	value
front	cpu-3-average_usage	value
front	cpu-3-cpu-idle	value
front	cpu-3-cpu-interrupt	value
front	cpu-3-cpu-nice	value
front	cpu-3-cpu-softirq	value
front	cpu-3-cpu-steal	value
front	cpu-3-cpu-system	value
front	cpu-3-cpu-user	value
front	cpu-3-cpu-wait	value
front	cpu-average_usage	value
front	cpufreq-average	value
front	cpufreq-cpufreq-0	value
front	cpufreq-cpufreq-1	value
front	cpufreq-cpufreq-2	value
front	cpufreq-cpufreq-3	value
front	disk-sda-disk_merged	read
front	disk-sda-disk_merged	write
front	disk-sda-disk_octets	read
front	disk-sda-disk_octets	write
front	disk-sda-disk_ops	read
front	disk-sda-disk_ops	write
front	disk-sda-disk_time	read
front	disk-sda-disk_time	write
front	entropy-entropy	entropy
front	interface-if_errors-eth0	rx
front	interface-if_errors-eth0	tx
front	interface-if_errors-eth1	rx
front	interface-if_errors-eth1	tx
front	interface-if_octets-eth0	rx
front	interface-if_octets-eth0	tx
front	interface-if_octets-eth1	rx
front	interface-if_octets-eth1	tx

front	interface-if_packets-eth0	rx
front	interface-if_packets-eth0	tx
front	interface-if_packets-eth1	rx
front	interface-if_packets-eth1	tx
front	memory-memory-buffered	value
front	memory-memory-cached	value
front	memory-memory-free	value
front	memory-memory-used	value
front	ping-ping-192.168.11.2	ping
front	ping-ping_droprate-192.168.11.2	value
front	ping-ping_stddev-192.168.11.2	value
front	processes-fork_rate	value
front	processes-java-ps_code	value
front	processes-java-ps_count	processes
front	processes-java-ps_count	threads
front	processes-java-ps_cputime	user
front	processes-java-ps_cputime	syst
front	processes-java-ps_data	value
front	processes-java-ps_disk_octets	read
front	processes-java-ps_disk_octets	write
front	processes-java-ps_disk_ops	read
front	processes-java-ps_disk_ops	write
front	processes-java-ps_pagefaults	minflt
front	processes-java-ps_pagefaults	majflt
front	processes-java-ps_rss	value
front	processes-java-ps_stacksize	value
front	processes-java-ps_vm	value
front	processes-ps_state-blocked	value
front	processes-ps_state-paging	value
front	processes-ps_state-running	value
front	processes-ps_state-sleeping	value
front	processes-ps_state-stopped	value
front	processes-ps_state-zombies	value
front	tcpconns-8080-local-tcp_connections-CLOSED	value
front	tcpconns-8080-local-tcp_connections-CLOSE_WAIT	value
front	tcpconns-8080-local-tcp_connections-CLOSING	value

front	tcpconns-8080-local-tcp_connections-ESTABLISHED	value
front	tcpconns-8080-local-tcp_connections-FIN_WAIT1	value
front	tcpconns-8080-local-tcp_connections-FIN_WAIT2	value
front	tcpconns-8080-local-tcp_connections-LAST_ACK	value
front	tcpconns-8080-local-tcp_connections-LISTEN	value
front	tcpconns-8080-local-tcp_connections-SYN_RECV	value
front	tcpconns-8080-local-tcp_connections-SYN_SENT	value
front	tcpconns-8080-local-tcp_connections-TIME_WAIT	value
front	vmem-direct_dma-vmpage_action-steal	value
front	vmem-direct_dma32-vmpage_action-steal	value
front	vmem-direct_movable-vmpage_action-steal	value
front	vmem-direct_normal-vmpage_action-steal	value
front	vmem-dma-vmpage_action-alloc	value
front	vmem-dma-vmpage_action-refill	value
front	vmem-dma-vmpage_action-scan_direct	value
front	vmem-dma-vmpage_action-scan_kswapd	value
front	vmem-dma32-vmpage_action-alloc	value
front	vmem-dma32-vmpage_action-refill	value
front	vmem-dma32-vmpage_action-scan_direct	value
front	vmem-dma32-vmpage_action-scan_kswapd	value
front	vmem-kswapd_dma-vmpage_action-steal	value
front	vmem-kswapd_dma32-vmpage_action-steal	value
front	vmem-kswapd_movable-vmpage_action-steal	value
front	vmem-kswapd_normal-vmpage_action-steal	value
front	vmem-movable-vmpage_action-alloc	value
front	vmem-movable-vmpage_action-refill	value
front	vmem-movable-vmpage_action-scan_direct	value
front	vmem-movable-vmpage_action-scan_kswapd	value
front	vmem-normal-vmpage_action-alloc	value
front	vmem-normal-vmpage_action-refill	value
front	vmem-normal-vmpage_action-scan_direct	value
front	vmem-normal-vmpage_action-scan_kswapd	value
front	vmem-vmpage_action-activate	value
front	vmem-vmpage_action-deactivate	value
front	vmem-vmpage_action-free	value
front	vmem-vmpage_faults	minflt

front	vmem-vmpage_faults	majflt
front	vmem-vmpage_io-memory	in
front	vmem-vmpage_io-memory	out
front	vmem-vmpage_io-swap	in
front	vmem-vmpage_io-swap	out
front	vmem-vmpage_number-active_anon	value
front	vmem-vmpage_number-active_file	value
front	vmem-vmpage_number-anon_pages	value
front	vmem-vmpage_number-anon_transparent_hugepages	value
front	vmem-vmpage_number-bounce	value
front	vmem-vmpage_number-dirtied	value
front	vmem-vmpage_number-dirty	value
front	vmem-vmpage_number-dirty_background_threshold	value
front	vmem-vmpage_number-dirty_threshold	value
front	vmem-vmpage_number-file_pages	value
front	vmem-vmpage_number-free_pages	value
front	vmem-vmpage_number-inactive_anon	value
front	vmem-vmpage_number-inactive_file	value
front	vmem-vmpage_number-isolated_anon	value
front	vmem-vmpage_number-isolated_file	value
front	vmem-vmpage_number-kernel_stack	value
front	vmem-vmpage_number-mapped	value
front	vmem-vmpage_number-mlock	value
front	vmem-vmpage_number-page_table_pages	value
front	vmem-vmpage_number-shmem	value
front	vmem-vmpage_number-slab_reclaimable	value
front	vmem-vmpage_number-slab_unreclaimable	value
front	vmem-vmpage_number-unevictable	value
front	vmem-vmpage_number-unstable	value
front	vmem-vmpage_number-vmscan_immediate_reclaim	value
front	vmem-vmpage_number-vmscan_write	value
front	vmem-vmpage_number-writeback	value
front	vmem-vmpage_number-writeback_temp	value
front	vmem-vmpage_number-written	value