Detecção de faltas: uma abordagem baseada no comportamento de processos

Cássio Martini Martins Pereira

Detecção de faltas: uma abordagem baseada no comportamento de processos

Cássio Martini Martins Pereira

Orientador: ***Prof. Dr. Rodrigo Fernandes de Mello***

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA.*

USP – São Carlos
Maio/2011

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

P436d

Pereira, Cássio Martini Martins
Detecção de faltas: uma abordagem baseada no comportamento de processos / Cássio Martini Martins Pereira; orientador Rodrigo Fernandes de Mello -- São Carlos, 2011.
109 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2011.

1. Tolerância a faltas. 2. Detecção de faltas. 3. Predição de faltas. 4. Aprendizado de máquina. 5. Tolerância a falhas. I. de Mello, Rodrigo Fernandes, orient. II. Título.

Agradecimentos

Agradeço a Deus por sempre iluminar e guiar meus caminhos.

A meus pais pelo apoio e paciência.

Ao professor Rodrigo Mello, por sua orientação presente e cuidadosa do trabalho.

Aos professores e funcionários do ICMC-USP, que auxiliaram direta e indiretamente o trabalho.

Aos colegas Eduardo Alves, Marcelo Albertini, Paulo Henrique Gabriel, Renato Ishii, Ricardo Rios e Vinicius Reis, pelas discussões que auxiliaram na realização do trabalho.

Às agências de fomento CNPq e FAPESP pelo auxílio financeiro.

Resumo

A diminuição no custo de computadores pessoais tem favorecido a construção de sistemas computacionais complexos, tais como aglomerados e grades. Devido ao grande número de recursos existentes nesses sistemas, a probabilidade de que faltas ocorram é alta. Uma abordagem que auxilia a tornar sistemas mais robustos na presença de faltas é a detecção de sua ocorrência, a fim de que processos possam ser reiniciados em estados seguros, ou paralisados em estados que não ofereçam riscos. Abordagens comumente adotadas para detecção seguem, basicamente, três tipos de estratégias: as baseadas em mensagens de controle, em estatística e em aprendizado de máquina. No entanto, elas tipicamente não consideram o comportamento de processos ao longo do tempo. Observando essa limitação nas pesquisas relacionadas, este trabalho apresenta uma abordagem para medir a variação no comportamento de processos ao longo do tempo, a fim de que mudanças inesperadas sejam detectadas. Essas mudanças são consideradas, no contexto deste trabalho, como faltas, as quais representam transições indesejadas entre estados de um processo e podem levá-lo a processamento incorreto, fora de sua especificação. A proposta baseia-se na estimação de cadeias de Markov que representam estados visitados por um processo durante sua execução. Variações nessas cadeias são utilizadas para identificar faltas. A abordagem proposta é comparada à técnica de aprendizado de máquina *Support Vector Machines*, bem como à técnica estatística *Auto-Regressive Integrated Moving Average*. Essas técnicas foram escolhidas para comparação por estarem entre as mais empregadas na literatura. Experimentos realizados mostraram que a abordagem proposta possui, com erro $\alpha = 1\%$, um F-Measure maior do que duas vezes o alcançado pelas outras técnicas. Realizou-se também um estudo adicional de predição de faltas. Nesse sentido, foi proposta uma técnica preditiva baseada na reconstrução do comportamento observado do sistema. A avaliação da técnica mostrou que ela pode aumentar em até uma ordem de magnitude a disponibilidade (em horas) de um sistema.

Abstract

The cost reduction for personal computers has enabled the construction of complex computational systems, such as clusters and grids. Because of the large number of resources available on those systems, the probability that faults may occur is high. An approach that helps to make systems more robust in the presence of faults is their detection, in order to restart or stop processes in safe states. Commonly adopted approaches for detection basically follow one of three strategies: the one based on control messages, on statistics or on machine learning. However, they typically do not consider the behavior of processes over time. Observing this limitation in related researches, this work presents an approach to measure the level of variation in the behavior of processes over time, so that unexpected changes are detected. These changes are considered, in the context of this work, as faults, which represent undesired transitions between process states and may cause incorrect processing, outside the specification. The approach is based on the estimation of Markov Chains that represent states visited by a process during its execution. Variations in these chains are used to identify faults. The approach is compared to the machine learning technique Support Vector Machines, as well as to the statistical technique Auto-Regressive Integrated Moving Average. These techniques have been selected for comparison because they are among the ones most employed in the literature. Experiments conducted have shown that the proposed approach has, with error $\alpha = 1\%$, an F-Measure higher than twice the one achieved by the other techniques. A complementary study has also been conducted about fault prediction. In this sense, a predictive approach based on the reconstruction of system behavior was proposed. The evaluation of the technique showed that it can provide up to an order of magnitude greater availability of a system in terms of uptime hours.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO

1

Introdução

1.1 Contextualização

Avanços tecnológicos reduziram os custos envolvidos na aquisição de computadores pessoais e máquinas de grande porte. Essas máquinas eram, inicialmente, projetadas para atender grandes volumes de requisições, sendo compostas por dezenas a centenas de elementos de processamento. O custo proibitivo envolvido no projeto e implementação dessas máquinas, aliado ao desenvolvimento tecnológico, motivou estudos e projetos de interconexão entre recursos computacionais de menor custo, os quais deram origem à área de sistemas distribuídos e, por sua vez, a tecnologias tais como aglomerados e grades (Foster et al., 2002; Kon e Goldman, 2008).

Aglomerados são caracterizados por computadores de arquitetura homogênea, interconectados em uma rede local de alta velocidade (Kon e Goldman, 2008). Grades são compostas por recursos de *hardware* e *software* heterogêneos e geograficamente distribuídos, interconectados por redes de diferentes características (Tanenbaum e Steen, 2007). Essas tecnologias são, inerentemente, voltadas para aplicações específicas, de grande porte e que buscam, essencialmente, por alto desempenho e alta disponibilidade. No entanto, para atender ambos requisitos, faz-se necessário prover consistência, segurança, escalonamento eficiente, tolerância a faltas, etc.

Entre esses aspectos, tolerância a faltas tem sido o foco de diversas pesquisas, tanto aplicadas em ambientes de aglomerado quanto de grades computacionais (Schroeder e Gibson, 2006; Filho et al., 2008; Feng e Lee, 2008). Para compreender as pesquisas existentes nessa área é importante, primeiramente,

definir seus principais termos: falta, erro e falha. Neste trabalho optou-se por empregar a taxonomia proposta por Gärtner (1999), a qual define esses termos como:

Falta: é uma transição inesperada entre estados do sistema;

Erro: é um estado fora da especificação do sistema;

Falha: é um erro em efeito, no qual a saída do sistema diverge de sua especificação.

Nesse contexto, diz-se que uma falta causa um erro, o qual pode, por sua vez, ocasionar uma falha no sistema. Dessa forma, tem-se a sequência ilustrada na Figura 1.1.

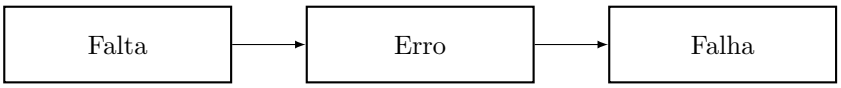


Figura 1.1: Sequência de passos para ocorrência de uma falha. Inicialmente uma transição inesperada (falta) leva o sistema a um estado errôneo (erro). O processamento incorreto nesse estado pode gerar um falha, que representa uma saída divergente da especificação do sistema.

Ambientes de grades executam, tipicamente, aplicações de longa duração, as quais são suscetíveis a perturbações, devido à diversidade e grande número de recursos computacionais existentes. O término inesperado dessas aplicações leva à perda das operações executadas, bem como à necessidade de tempo sobressalente para executar tais tarefas novamente.

Entre esses ambientes, grades computacionais apresentam maior nível de heterogeneidade, além de outros aspectos que tornam ainda mais complexo o gerenciamento e manutenção de sua disponibilidade, tais como: operações desconectadas, segurança, alto grau de distribuição de recursos, etc. Esses fatores tornam aplicações em grades mais propensas a falhas resultantes de elementos independentes ou da interação entre eles (Medeiros et al., 2003).

Para exemplificar fatores motivadores de falhas, considere, por exemplo, um sistema que executa em um ambiente de grade computacional. A falha individual de um processador, módulo de memória, interface de rede ou, por exemplo, disco rígido, pode causar o término inesperado do sistema.

Em outra situação, considere uma aplicação BSP (*Bulk Synchronous Parallel*) (Cheatham et al., 1995) executando sobre um subconjunto das máquinas de uma grade. Esse tipo de aplicação permite a comunicação entre diferentes processos, ao contrário de *bag-of-tasks*, em que os processos trabalham de forma independente. Caso haja particionamento de rede, os computadores envolvidos no processamento podem perder a capacidade de comunicação e, consequentemente, gerar saídas incorretas, que divergem da especificação do sistema.

Frente a essas questões, a detecção é o primeiro passo para tornar um sistema tolerante a faltas. A partir da detecção, medidas apropriadas podem ser tomadas, tais como parar os processos em execução em um estado seguro (*fail-safe*), ou reiniciá-los em um estado anteriormente seguro, por meio de *checkpointing/restart* (Gärtner, 1999).

Segundo Avizienis et al. (2004), a detecção de faltas é realizada por meio de mecanismos de sistema que verificam a corretude do serviço oferecido. Tais mecanismos apresentam dois tipos de problemas: 1) avisar que houve uma falta quando de fato isso não ocorreu (falso positivo) e 2) não avisar que uma falta ocorreu, quando de fato foi o caso (falso negativo).

Diversas abordagens têm buscado detectar faltas. Elas podem ser divididas em três frentes básicas: abordagens baseadas em *heartbeats*, abordagens estatísticas e as baseadas em aprendizado de máquina. Foram inclusas nessa lista as abordagens baseadas em *heartbeats* por serem as mais comuns na literatura, entretanto, conforme será discutido, elas são capazes apenas de detectar falhas.

As abordagens baseadas em *heartbeats* foram concebidas para utilização em sistemas distribuídos. Elas são basicamente subdivididas em ativas e passivas. Técnicas ativas funcionam por meio de um monitor que envia mensagens de controle periódicas a processos monitorados (*Pull*), para verificar se eles estão ativos ou não. Já em técnicas passivas, o monitor recebe, periodicamente, informações sobre o estado dos processos monitorados (*Push*). Caso um intervalo de tempo Δt transcorra sem o recebimento de um "*heartbeat*", o processo monitorado passa a ser considerado falho.

Na área de detecção baseada em *heartbeats* existem trabalhos teóricos clássicos, como o de Fischer et al. (1985) que provaram a impossibilidade de consenso (Lamport et al., 1982) em um sistema distribuído assíncrono com um processo defeituoso. Esse trabalho motivou pesquisas na área, as quais buscaram adicionar algumas restrições ao problema para torná-lo solucionável. Nesse contexto, Chandra et al. (1996) propuseram o detector de falhas mais fraco possível (que faz menos suposições e restrições ao problema) para resolver consenso. Além de trabalhos teóricos, é possível citar também aplicações dessas abordagens. Hayashibara et al. (2002) analisaram problemas relacionados a detectores de falhas para grades, tais como explosão do número de mensagens de *heartbeat*. Os autores concluíram que modelos baseados em *Gossiping* (van Renesse et al., 1998) podem auxiliar a diminuir o número de mensagens enviadas. Filho et al. (2008) adotaram o modelo probabilístico Accrual para projetar um detector de falhas para grades computacionais oportunistas.

Uma desvantagem das técnicas puramente baseadas em *heartbeats* é que

elas somente detectam falhas, ou seja, situações em que os processos monitorados já falharam e pararam de responder. Uma análise mais detalhada dos processos monitorados pode fornecer mais indícios sobre a situação de cada um deles. Nesse contexto, foram propostas abordagens que visam aprender características de um processo durante sua execução. Essa abordagens utilizam conceitos estatísticos e de aprendizado de máquina.

Diversos trabalhos na área de tolerância a faltas têm empregado técnicas estatísticas a fim de detectar eventos raros, tratados como faltas. No contexto de *software rejuvenation* (Huang et al., 1995), busca-se pelo momento ideal para reiniciar um processo, visando amenizar faltas causadas por defeitos de *software*, como por exemplo a exaustão de memória decorrente de "vazamentos" (*memory leaks*). Nessa linha, Li et al. (2002) construíram modelos ARMA (*Auto-Regressive Moving Average*), a partir de dados coletados de um servidor *Web*, a fim de detectar a exaustão de recursos e encontrar o melhor momento para reiniciar o servidor. Xue et al. (2007) observaram que modelos auto-regressivos (AR), de médias móveis (MA), auto-regressivos de médias móveis (ARMA) e auto-regressivos integrados de médias móveis (ARIMA), podem ser utilizados para modelar o perfil de sistemas, a fim de detectar momentos de exaustão de recursos.

Outros trabalhos têm utilizado técnicas de aprendizado de máquina. Parte desses trabalhos está voltada para detecção de faltas, enquanto outra parte visa apenas detectar falhas. É importante observar que uma falta leva a uma falha, logo detectar faltas auxilia na prevenção de falhas.

Turnbull e Alldrin (2003) utilizaram redes neurais artificiais RBF (*Radial Basis Function*), devido a seu treinamento e classificação eficientes, para predizer faltas em *hardware*, baseando-se em dados de sensores de placas mãe.

Ning et al. (2006) empregaram redes neurais em conjunto com conceitos *Fuzzy* e *Wavelets* para modelar dados de desempenho de sistema e predizer o envelhecimento de *software* em servidores de aplicação, obtendo bons resultados.

Hoffmann et al. (2007) analisaram as técnicas *Support Vector Machines* (SVM), *Radial Basis Functions* (RBF), Regressão Linear Multivariada e *Universal Basis Functions* (UBF) e concluíram que, entre essas, SVM's apresentam o menor erro para previsões de curto e longo prazo do tempo de resposta do servidor Apache[1], auxiliando a detectar momentos de possíveis faltas.

El-Shishiny et al. (2008) investigaram o uso de redes neurais artificiais Multi-Layer Perceptron para minerar e predizer padrões de envelhecimento de *software*.

Os trabalhos existentes em ambas subáreas mencionadas (estatística e

[1] http://httpd.apache.org. Último acesso: 25/01/2011.

aprendizado de máquina) não levam em consideração diversas informações sobre os processos computacionais analisados, que estão sujeitos a faltas. São desconsideradas, por exemplo, as chamadas de sistema realizadas, juntamente com atributos e retorno dessas funções, o que resulta na perda de informações relevantes para a detecção de eventos anômalos (faltas).

Observando as limitações das pesquisas relacionadas, tanto em abordagens baseadas em *heartbeats*, quanto nas baseadas em estatística e aprendizado de máquina, foi formulada a hipótese deste trabalho, de que a variação do comportamento de um processo, em termos das chamadas de sistema realizadas, bem como dos valores associados aos seus argumentos, pode fornecer indícios relevantes para a detecção de faltas. Espera-se que, ao empregar essas informações, seja possível ter maior eficácia do que as técnicas existentes de detecção de faltas.

A justificativa para a hipótese assumida é dada em função de pesquisas na área de escalonamento em grades computacionais, as quais utilizaram o conhecimento sobre processos e obtiveram bons resultados (Devarakonda e Iyer, 1989; Kao et al., 1993; Dodonov e de Mello, 2007; de Mello et al., 2009). Espera-se que ao aplicar esse conhecimento na área de tolerância a faltas obtenha-se, também, bons resultados.

1.2 Objetivo

As limitações encontradas nas técnicas empregadas para detecção de faltas (Hoffmann et al., 2006; Xue et al., 2007), bem como os bons resultados oriundos do emprego do comportamento de processos na área de escalonamento (Kao et al., 1993; de Mello et al., 2009), motivaram o uso desse conhecimento para detectar faltas durante a execução de processos.

O objetivo deste trabalho consiste em desenvolver uma abordagem de detecção de faltas que considere variações no comportamento de processos ao longo do tempo. Essas variações são caracterizadas por mudanças nos argumentos e retornos de chamadas de sistema, bem como por parâmetros de desempenho do ambiente no qual o processo é executado. Espera-se que essa abordagem detecte faltas de maneira mais eficaz do que demais técnicas existentes.

Com essa finalidade, conceitos de cadeias de Markov e Entropia foram estudados, a fim de identificar estados que um processo visita durante sua execução e medir a variação nesse comportamento. Com essa modelagem espera-se obter uma melhor representação dos estados que compõem processos.

1.3 Organização

O próximo capítulo apresenta conceitos relacionados a tolerância a faltas, a taxonomia adotada nesta pesquisa e trabalhos relacionados à detecção de faltas. No Capítulo 3, conceitos relacionados a redes neurais artificiais RBF, cadeias de Markov e Teoria da Informação são apresentados. Em seguida, no Capítulo 4, apresenta-se a abordagem de detecção de faltas proposta. O Capítulo 5 discute os experimentos realizados, segundo a abordagem proposta. Para complementar o trabalho, no Capítulo 6 é apresentado um estudo sobre predição de faltas, bem como a abordagem preditiva proposta. Por fim, no Capítulo 7, são apresentadas as conclusões.

CAPÍTULO

2

Tolerância a faltas

2.1 Considerações iniciais

Com o intuito de formalizar os principais conceitos relativos à área de tolerância a faltas, este trabalho adota duas propostas de taxonomia, elaboradas por Gärtner (1999) e Avizienis et al. (2004). Ambas são apresentadas a seguir, bem como uma discussão de trabalhos relacionados.

2.2 Taxonomia

A literatura apresenta duas taxonomias comumente adotadas para definir termos e técnicas utilizadas na área de tolerância a faltas. A primeira, proposta por Gärtner (1999), aborda diversos conceitos e os aplica em um cenário distribuído. A segunda, proposta por Avizienis et al. (2004), define conceitos sobre as áreas de tolerância a faltas e segurança computacional.

Gärtner (1999) e Arora e Gouda (1993) analisaram terminologias relativas à área de tolerância a faltas e observaram definições distintas e irregulares para os mesmos aspectos abordados. Gärtner (1999) destaca a relevância em unificar e definir, claramente, termos relativos a essa área, pois simplificariam a compreensão de seus problemas inerentes, bem como a forma de tratá-los. Esse aspecto o motivou a definir diversos termos, tais como falta, erro e falha. Para instanciar tais termos esse autor considera um sistema distribuído assíncrono, pois é o modelo mais próximo daqueles encontrados em ambientes distribuídos reais. Além disso, o modelo assíncrono define menos suposições e restrições que o sistema deve respeitar. Dessa forma, resultados aplicáveis

a ele também podem ser empregados ao modelo síncrono.

Segundo esse modelo, um conjunto de processadores V, interconectados por canais de comunicação E, encontra-se organizado segundo um grafo $G = (V, E)$. Cada elemento $v \in V$ apresenta capacidade computacional[1] e carga de trabalho distintas. Cada canal $e \in E$, onde $e = \{v_i, v_j\}$, apresenta atrasos arbitrários e largura de banda distinta para comunicação entre pares de vértices. Considere que um conjunto de processos P, composto por $|P| > 1$ elementos, foi alocado sobre um subconjunto $V' \subset V$, onde $|V'| > 1$. Processos em P comunicam-se por meio de troca de mensagens, utilizando os canais de comunicação em E. A fim de abordar um evento específico, considere um processo p_i que encaminha uma mensagem m para p_j, e que há um arco ou conjunto de arcos que os conectam, no instante t. Dadas as capacidades distintas dos elementos em V, dos canais em E e de suas respectivas cargas, nada se pode supor ou garantir sobre o intervalo exato de tempo δt necessário para a entrega de m para p_j, processamento da mensagem m, geração e envio de uma resposta.

Sobre esse modelo, Gärtner (1999) e Avizienis et al. (2004) formalizam os termos falta, erro e falha. Falta representa uma transição indesejada entre estados que compõem o comportamento de um sistema. Um erro é causado por uma falta, que leva o sistema a um estado incorreto. Um erro pode levar a uma falha, a qual indica que o sistema divergiu de seu comportamento especificado.

A fim de melhor compreender esses termos, considere a Figura 2.1 onde uma máquina de estados finitos é apresentada, a qual modela o comportamento de um processo. O conjunto de estados é formado por $E = \{E_1, E_2, E_3\}$, os quais apresentam transições associadas. Essas transições podem ser acionadas por uma entrada específica, dada pelo alfabeto $\{0, 1\}$, ou por quaisquer entradas inesperadas contidas no conjunto de transições $T = \{t_0, t_1\}$. Considere, ainda, que os estados E_1 e E_2 são corretos e, portanto, seguem a especificação do processo. E_3 representa um estado errôneo, o qual diverge da especificação.

A transição do estado E_1 para E_2 pode ocorrer da forma esperada, ou seja, dada pela entrada 0 ou por uma entrada $t_0 \in T$. Caso ocorra t_0, a máquina muda para um estado correto, no entanto, diz-se que essa transição é motivada por um evento inesperado. A esse evento dá-se o nome de falta. Considere, agora, uma transição do estado E_2 para E_3 em função da entrada t_1. Esse evento indica uma falta, conforme descrito anteriormente, contudo, nesta oca-

[1] O termo capacidade se refere ao desempenho dos recursos envolvidos no vértice. Sendo o vértice um processador, a capacidade se refere ao desempenho em termos de instruções processadas por segundo. Sendo o vértice um computador, o termo envolve o desempenho de processamento, memória, disco, etc.

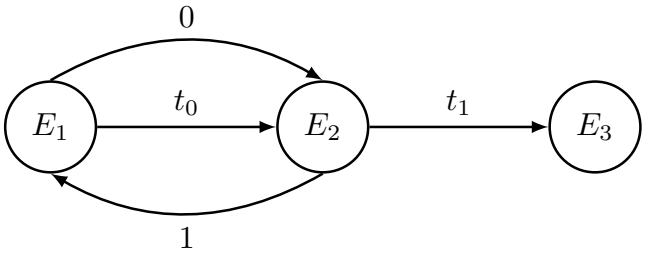


Figura 2.1: Máquina de estados finitos. E_1 e E_2 são estados corretos. E_3 representa um estado incorreto, ou não especificado. A transição inesperada t_1 (falta) pode levar o sistema do estado correto E_2 para E_3 (erro), o qual poderá gerar uma falha.

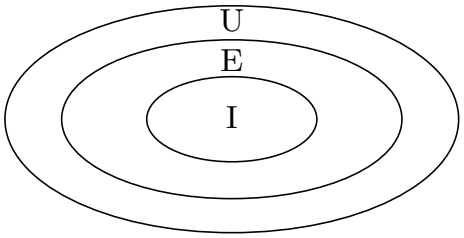


Figura 2.2: Conjunto de estados de um processo. U representa o universo de possíveis estados. E representa o conjunto de estados previstos e alcançáveis pelo processo (corretos ou não). I é o conjunto de estados corretos (logo esperados), também chamados de invariantes.

são a máquina foi levada a um estado errôneo, onde ocorrerá processamento e geração de resposta incorreta. O estado E_3 representa um erro, enquanto o resultado gerado por seu processamento, o qual diverge do modelo especificado, indica uma falha.

Gärtner (1999) define os estados especificados de um sistema como invariantes. No processo modelado anteriormente, as invariantes são dadas pelo conjunto $I = \{E_1, E_2\}$, em que $I \subset E$. Há, contudo, um conjunto universo U, ilustrado na Figura 2.2, que contém todos os elementos em I e demais estados que podem ser alcançados por eventos inesperados $t \in T$. Portanto, tem-se $I \subset E \subset U$, onde o resultado de $U - I$ contém todos estados não especificados pelo sistema, ou errôneos, que podem levar à ocorrência de falhas. O subconjunto dado por $N = E - I$ contém todos os estados errôneos previstos e alcançáveis por transições em T. Pode-se, portanto, dizer que o sistema é T-tolerante, pois prevê e trata cada uma das faltas em T.

Observa-se, por essa formalização, que uma falta pode levar a estados errôneos e, consecutivamente, a falhas. Portanto, nesse contexto, a origem ou causa de um erro ou falha é uma falta, o que motiva seu estudo.

Após formalizar termos relacionados à área de tolerância a faltas, faz-se necessário caracterizar duas propriedades que permitem classificar abordagens para o tratamento de faltas. Essas propriedades, apresentadas por Gärtner (1999), são *safety* e *liveness*. A primeira propriedade (*safety*) é atendida por sistemas que apresentam um conjunto de w execuções, dado por $X = \{x_0, x_1, \ldots, x_{w-1}\}$, em que cada execução é composta por uma sequência

Tabela 2.1: Classificação das Formas de Tolerância a Faltas (Gärtner, 1999)

	Live	***Not Live***
Safe	*Masking*	*Fail Safe*
Not Safe	*Non-Masking*	*None*

de transições, $x_n = (x_{n0}, x_{n1}, \ldots)$, que não violam invariantes em I. A segunda, ou *liveness*, garante que o sistema progride sua execução.

Em termos formais, pode-se caracterizar ambas propriedades de acordo com o conjunto de estados visitados durante a execução de um sistema. *Safety* garante que somente estados pertencentes ao conjunto I serão visitados durante a execução. *Liveness*, por sua vez, garante que, eventualmente, o sistema transiciona entre quaisquer estados em U.

Essas propriedades permitem definir os quatro níveis de tolerância a faltas existentes (Gärtner, 1999) (Tabela 2.1): *Masking*, *Fail Safe*, *Non-Masking* e *None*. A primeira forma, *Masking*, caracteriza sistemas que progridem suas execuções de forma segura. Nesse caso, faltas são mascaradas pelo sistema. O segundo tipo, ou *Fail Safe*, garante que o sistema para sua execução em um estado seguro. Essa abordagem foi adotada, por exemplo, na construção do sistema de controle terrestre do foguete espacial Ariane 5 (Dega, 1996). Esse sistema foi projetado para mascarar faltas de um único componente, e parar em um estado seguro no caso de duas ou mais faltas sucessivas. Para o caso de múltiplas faltas, a continuidade do lançamento do foguete (*liveness*) seria de menor importância que a segurança (*safety*). Na situação *Non-Masking*, o sistema progride, porém pode, eventualmente, visitar estados incorretos. No entanto, há pesquisas que buscam por algoritmos, denominados *self-stabilizing* (Dijkstra, 1974; Sridharan et al., 2008), capazes de retornar a um estado correto. A quarta e última forma, denominada *None*, não garante progressão nem segurança, portanto não oferece nenhum tipo de tolerância a faltas.

Em geral, pesquisas que privilegiam a propriedade *safety* estão relacionadas com abordagens para detecção de faltas (Ebnenasir et al., 2008; Bonakdarpour e Kulkarni, 2008; Lu e Halang, 2009), enquanto aquelas que priorizam *liveness* têm foco na recuperação do sistema na presença de faltas (Sridharan et al., 2008; Balazinska et al., 2008; Turau e Weyer, 2009). No entanto, ambos ramos de pesquisa utilizam-se de técnicas de detecção de faltas. Enquanto é necessário determinar se uma falta ocorreu para garantir que o sistema não viole condições de segurança (*safety*), isso também é necessário para que o sistema possa progredir na presença de faltas (*liveness*) e, eventualmente, recuperar-se delas. Isso motivou pesquisas na área de detecção, as

quais abordaram a questão de diversas maneiras.

2.3 Detecção de faltas

Grande parte dos trabalhos relacionados à área de tolerância a faltas propõe ou utiliza técnicas para detecção de eventos inesperados (faltas). Esta seção apresenta trabalhos relacionados que adotam tais abordagens.

2.3.1 Conceitos e detectores baseados em heartbeats

Diversos trabalhos na área de tolerância a faltas (Hayashibara et al., 2002; Nurmi et al., 2005; Filho et al., 2008), e mais especificamente de grades computacionais, têm abordado a questão de detecção. Alguns desses trabalhos visam somente a detecção de falhas, enquanto outros buscam, também, detectar faltas.

Muitos desses trabalhos utilizam mensagens de controle, denominadas *heartbeats*, para inquirir um processo remoto sobre seu estado. Caso o processo não responda à mensagem de controle em um intervalo de tempo Δt, considera-se que ele falhou. Trabalhos pesquisados que adotam essa abordagem são apresentados a seguir.

Hayashibara et al. (2002) identificam problemas críticos na implementação de um serviço escalável de detecção de falhas em grades computacionais. Nesse contexto, os autores evidenciam a necessidade de projetar protocolos específicos capazes de operar e considerar diferentes circunstâncias, tais como: distribuição geográfica de recursos, níveis de assincronia, taxa de atrasos na entrega de mensagens, probabilidade de perda de mensagens e organização dinâmica dos elementos.

Os autores reportam seis problemas que consideram importantes para a construção de detectores de falha escaláveis, levando em consideração as circunstâncias descritas anteriormente. Esses problemas são: 1) explosão de mensagens: apesar do alto número de componentes que devem ser monitorados, o detector deve evitar *flooding* de avisos de falha na rede; 2) escalabilidade: o detector deve ser capaz de monitorar, de maneira eficiente, um grande número de recursos distribuídos. Ele precisa rapidamente detectar falhas, enquanto minimiza o número de falsos positivos; 3) perda de mensagens: o detector deve ser sensível a mudanças na rede. Em redes globais, perdas de mensagens ocorrem com alta probabilidade, como resultado de falhas transientes ou da atenuação de sinais em função de ruídos. Mensagens podem sofrer atrasos arbitrários. Esses eventos podem, incorretamente, ser reportados como falhas, gerando falsos positivos; 4) flexibilidade: o serviço de detecção de falhas deve adaptar-se a diferentes aplicações e requisitos de

monitoramento de recursos do sistema, pois isso tende a reduzir o número de mensagens geradas; 5) dinamismo: uma grade é um ambiente altamente dinâmico, com componentes que entram e saem constantemente, padrões de uso variáveis, mudanças na topologia das redes, etc. Detectores precisam adaptar-se à reconfiguração da grade. 6) segurança: detectores de falhas podem ser usados para ataques de negação de serviço (*Denial of Service* – DoS), pois tendem a gerar mensagens de suspeita que sobrecarregam recursos.

No modelo de Hayashibara et al. (2002), cada processo tem acesso a um módulo local para a detecção de falhas. Cada módulo monitora um subconjunto de processos do sistema. O módulo mantém uma lista de processos suspeitos de terem falhado. Os autores definem dois métodos para o funcionamento do módulo detector de falha:

Push: nesse método, os componentes monitorados são ativos e o monitor (detector de falhas) é passivo. O componente monitorado envia periodicamente mensagens, chamadas de *heartbeats*, ao monitor. Após um período de *timeout* Δt sem receber *heartbeats*, considera-se que o processo monitorado falhou;

Pull: nesse método, os componentes monitorados são passivos, enquanto o detector de falhas é ativo. O monitor envia mensagens do tipo "você está vivo?" periodicamente aos componentes monitorados. Respostas positivas indicam que eles estão ativos. A ausência de resposta de um processo monitorado indica que ele falhou.

Um problema com ambos métodos (*Push*/*Pull*) é que processos lentos, que demoram para responder ou que enfrentam maior latência na rede para o envio de seus *heartbeats*, podem ser considerados falhos sem terem falhado na realidade.

Os autores identificam que detectores baseados em *Gossip* (van Renesse et al., 1998), ou seja, que basicamente atribuem uma probabilidade ao envio de mensagens via rede, podem solucionar problemas de explosão de mensagens (Problema 1), perda de mensagens (Problema 3) e dinamismo (Problema 5). É possível identificar duas versões desses protocolos: *Basic Gossiping* e *Multi-Level Gossiping*.

No *Basic Gossiping*, o módulo detector de falhas (DF) fica residente em cada *host* da rede. Ele mantém uma lista com cada DF que conhece. A cada entrada nessa lista está associado um contador de *heartbeats*. Cada DF seleciona outro aleatoriamente (sem levar em conta a topologia da rede) e envia sua lista após incrementar o seu contador de *heartbeat*. O DF receptor unifica (faz *merge*) sua lista local com a lista recebida e adota o maior contador de *heartbeat* para cada entrada da lista.

Ocasionalmente cada membro faz um *broadcast* da sua lista, para recuperação de perdas causadas por eventuais partições na rede. Se o contador de um *host* A, mantido na lista de um *host* B, não for incrementado após um período de *timeout* Δt, B suspeita que A falhou.

Para adaptar-se a redes de larga escala foi proposta uma variante do protocolo básico, chamada de *Multi-Level Gossiping*. Esse modelo usa a estrutura de domínios da Internet e subredes para mapear detectores em diferentes níveis. A maior parte das mensagens trocadas entre detectores são enviadas pelo protocolo básico dentro de uma mesma subrede, algumas entre subredes e poucas entre domínios.

Protocolos *gossip*, no entanto, são ruins quando uma grande porcentagem de componentes falha ou fica particionada, pois o detector pode demorar muito para receber o aviso de que um componente falhou. Os autores concluem, a partir desse estudo, que abordagens híbridas podem solucionar os seis problemas identificados (explosão de mensagens, escalabilidade, perda de mensagens, flexibilidade, dinamismo e segurança).

Em outro estudo sobre ambientes de larga escala, Nurmi et al. (2005) modelam a disponibilidade de máquinas em sistemas geograficamente distribuídos. Os autores utilizam três *data sets* para realizar seu estudo. O primeiro contém informações sobre o tempo entre reinícios de computadores do Laboratório de Instrução em Ciências da Computação (CSIL), coletados na Universidade da Califórnia, Santa Bárbara (UCSB). O segundo *data set* contém dados sobre o tempo de execução de processos, observados no sistema Condor da Universidade de Wisconsin durante um período de dois meses. O terceiro, por sua vez, consiste de dados coletados por Long et al. (1995), que identificaram 1170 *hosts* conectados à Internet em 1995. Os dados são compostos por respostas desses computadores a consultas do rpc.statd – processo associado ao *Network File System* (NFS). Respostas são utilizadas como mensagens de *heartbeat*. A ausência de resposta indica falha.

Com base em estudos estatísticos dos três *data sets*, os autores identificaram que o histórico de disponibilidade das máquinas, obtido ao longo do tempo, influencia seu comportamento futuro. Os autores propuseram a adoção das distribuições de probabilidade *Weibull* e Hiperexponencial para modelagem da disponibilidade, devido a seus aspectos *non-memoryless*, e concluíram que essas são mais adequadas para representar os *data sets* analisados. Essas distribuições podem ser utilizadas em simuladores para melhor modelar ambientes reais.

Nesse contexto de simulação, Caminero et al. (2007) introduzem mecanismos para simular e detectar falhas no *toolkit* GridSim, um simulador de grades computacionais. O GridSim modela recursos computacionais hetero-

gêneos com desempenho variável, políticas de escalonamento baseadas em tempo ou espaço e serviços de rede diferenciados. O modelo construído pode, posteriormente, ser utilizado para simulação baseada em eventos.

No simulador proposto por Caminero et al. (2007), o método escolhido para detecção de falhas de recursos foi o modelo *Pull*, oposto ao *Push*, como definido por Hayashibara et al. (2002). Esse modelo foi escolhido pelo fato de que no modelo *Push* é possível que um *heartbeat* perdido seja interpretado como a falha de um recurso ou a perda de conexão de rede, o que pode ocasionar falsos positivos.

Os autores propõem a utilização de uma distribuição Hiperexponencial para modelar o tempo entre ocorrências de falhas na grade, pois essa mostrou-se adequada na representação da disponibilidade de recursos, como proposto por Nurmi et al. (2005). Caso o simulador gere uma falha, módulos de detecção são responsáveis por solicitar o reescalonamento e migração dos processos envolvidos.

Caminero et al. (2007) concluem que a ocorrência de falhas, bem como sua detecção, são aspectos importantes a serem pesquisados no contexto de grades computacionais, dada sua frequente ocorrência. Isso motiva, também, o estudo de detecção de faltas, dado que uma falta é a causa original de uma falha. Nesse sentido, ferramentas de simulação auxiliam o estudo do comportamento de diferentes abordagens, uma vez que a disponibilidade de recursos computacionais reais é custosa e de pouco acesso para análise de falhas.

Ainda no contexto de grades, Filho et al. (2008) adotam o modelo Accrual (Hayashibara et al., 2004) para projetar um detector de falhas para grades computacionais oportunistas. Segundo esse modelo, detectores utilizam o tempo entre chegadas de mensagens de *heartbeat*, provenientes dos processos monitorados, para compor uma distribuição de probabilidade, a qual caracteriza o tempo de resposta desses processos. Com base nessa distribuição e no tempo decorrido desde a chegada do último *heartbeat*, os detectores geram como saída um valor contínuo, que representa o nível de suspeita sobre a falha de um processo.

Segundo os autores, o diferencial do modelo Accrual é que sua saída não representa um valor binário de suspeita, indicando apenas se o processo monitorado falhou ou não, mas sim um valor contínuo, que permite associar diferentes estratégias de tolerância, de acordo com o nível correspondente. Dessa maneira, espera-se que o detector seja mais flexível do que os puramente binários.

O problema com a abordagem adotada pelos trabalhos apresentados, notadamente o emprego de mensagens de controle (*heartbeats*), é que ela não considera informações sobre o estado dos processos monitorados. Essa aborda-

gem consegue detectar apenas falhas, ou seja, instantes em que um processo já sofreu uma falta, visitou um estado errôneo e divergiu de sua especificação, a ponto de não responder mais. Nesse contexto, foram propostas abordagens que levam em conta uma granulosidade mais fina de informações sobre os processos. Essas abordagens utilizam, muitas vezes, conceitos estatísticos e de aprendizado de máquina.

2.3.2 Conceitos e detectores baseados em estatística

Vários trabalhos na área de tolerância a faltas têm empregado técnicas estatísticas para modelar o comportamento do ambiente em que um processo é executado. Variáveis de desempenho, tais como memória física disponível, número de chaveamentos de contexto por segundo e uso de CPU, são utilizadas para modelagem, porém sem levar em conta características específicas de cada processo. Variações em medições do ambiente são então empregadas para detectar faltas. Essas técnicas, em geral, utilizam conceitos de séries temporais para analisar observações coletadas no tempo.

Xue et al. (2007) analisaram diversos trabalhos relacionados a detecção de faltas e que empregam conceitos estatísticos de séries temporais. Os autores observaram que modelos auto-regressivos (AR), de médias móveis (MA), auto-regressivos de médias móveis (ARMA) e auto-regressivos integrados de médias móveis (ARIMA) têm sido utilizados em diversas pesquisas para modelar o perfil de sistemas.

Entre os modelos disponíveis de séries temporais, Xue et al. (2007) observaram que a maior parte dos trabalhos emprega o modelo ARMA. Para descrever esse modelo é necessário, anteriormente, apresentar conceitos sobre séries temporais.

Segundo Morettin e Toloi (2006), para descrever séries temporais utilizam-se modelos baseados em processos estocásticos, ou seja, processos probabilísticos. Um processo estocástico é definido como uma família $Z = \{Z(t), t \in T\}$, em que T é um conjunto arbitrário (normalmente adotado como $\mathbb{Z}$ dos inteiros ou $\mathbb{R}$ dos reais) e $\forall t, Z(t)$ é uma variável aleatória. Disso decorre que $Z(t)$ é uma variável aleatória definida sobre o espaço amostral Ω, e portanto é uma função de duas variáveis, $Z(t, \omega), t \in T, \omega \in \Omega$. A Figura 2.3 ilustra um exemplo de processo. Para um dado ω fixo, tem-se uma função em t, que é chamada de realização ou trajetória do processo, ou ainda, série temporal.

Designando-se as séries temporais de $Z(t, \omega)$ por $Z^{(1)}(t), Z^{(2)}(t), \ldots$, tem-se que cada $Z^{(j)}(t)$ é uma função em t não aleatória (com ω fixo) e $\forall t \in T, Z^{(j)}(t) \in \mathbb{R}$. Simplificando a representação de uma série temporal $Z^{(j)}(t)$ por $Z(t)$, ao conjunto de valores $\{Z(t), t \in T\}$, dá-se o nome de espaço de estados $\mathcal{E}$ do processo estocástico. Os valores $Z(t)$ são chamados de estados. Caso T seja

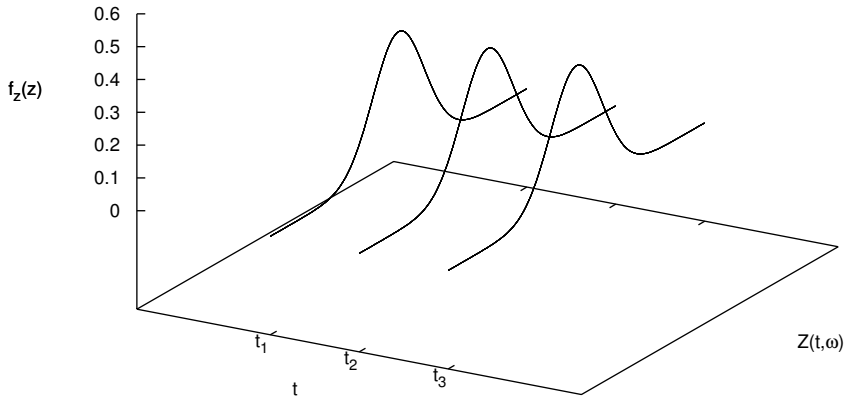


Figura 2.3: Representação de um processo estocástico. Para cada t tem-se uma variável aleatória $Z(t, \omega)$, com uma função densidade probabilidade (f.d.p.) $f_Z(z)$. É possível que a f.d.p. seja diferente para quaisquer t_1 e t_2, $t_1 \neq t_2$. Entretanto, comumente tem-se uma mesma f.d.p. para todo $t \in T$ (Morettin e Toloi, 2006).

finito ou infinito numerável, por exemplo quando T assume valores em $\mathbb{Z}$, diz-se que o processo é de parâmetro discreto. Caso T possa assumir quaisquer valores em $\mathbb{R}$, diz-se que o processo é de parâmetro contínuo. Da mesma forma, $\mathcal{E}$ pode ser discreto ou contínuo. Por exemplo, caso $Z(t)$ seja o número de processos na fila de execução de uma máquina, $\mathcal{E} \subset \mathbb{Z}$, já se $Z(t)$ representa a temperatura de um disco rígido, por exemplo, $\mathcal{E} \subset \mathbb{R}$.

De acordo com Morettin e Toloi (2006), "uma série temporal é qualquer conjunto de observações ordenadas no tempo". Por exemplo, valores da precipitação atmosférica anual em uma cidade, índices da bolsa de valores, número médio anual de manchas solares, etc. Uma das possíveis maneiras de analisar essas séries é estudá-las no domínio temporal, utilizando modelos paramétricos (que possuem número finito de parâmetros), tais como os modelos ARMA e ARIMA.

Segundo Weedon (2003), outra possível abordagem é o estudo no domínio das frequências, utilizando técnicas que não dependem da natureza dos dados. Por essa razão, essas técnicas são chamadas de não paramétricas e envolvem, basicamente, uma aproximação de uma função do tempo por combinações lineares de senóides, com coeficientes compostos pelas transformadas de Fourier discretas da série (Morettin e Toloi, 2006).

Pode-se descrever uma série temporal como um vetor $Z(t)$, de ordem $r \times 1$, em que t é um vetor $p \times 1$. Por exemplo:

$$Z(t) = [Z_1(t), Z_2(t), Z_3(t)]' \tag{2.1}$$

Na série 2.1, os componentes poderiam denotar consumo de CPU (Z_1), memória física disponível (Z_2) e número de processos na fila de execução (Z_3), de um computador identificado por um determinado endereço IP (*Internet Protocol*) em um momento específico, ou seja, $t = (tempo, ip)$. Neste exemplo, a série considerada é multivariada com $r = 3$ e multidimensional, com $p = 2$.

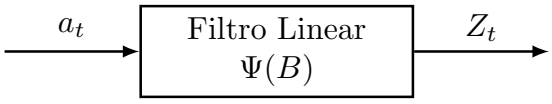


Figura 2.4: Representação de um filtro linear. A entrada é o ruído branco a_t, que é submetido à função de transferência $\Psi(B)$, cuja saída é Z_t (Morettin e Toloi, 2006).

De acordo com Morettin e Toloi (2006), pode-se construir modelos probabilísticos para representar essas séries no domínio do tempo. Esses modelos devem ser simples e parcimoniosos, ou seja, devem utilizar o menor número de parâmetros para representar adequadamente a série. Tais metas seguem o princípio da Navalha de Occam (Mitchell, 1997), o qual sugere que ao analisar hipóteses competidoras, quando iguais em outros aspectos, deve-se selecionar a que faz menos suposições, requer menos objetos, e ainda assim, explica de maneira suficiente o problema.

Além da busca por modelos mais simples, geralmente faz-se a suposição de que a série estudada é estacionária, ou seja, de que ela se comporta aleatoriamente no tempo, orbitando em torno de uma média e variância constantes. Entretanto, na prática, a maioria das séries apresenta não estacionariedade (Morettin e Toloi, 2006). Por exemplo, séries que orbitam em torno de uma reta apresentam o que se chama de tendência linear. Por outro lado, algumas séries exibem comportamento explosivo, como é o caso de uma série que denota o crescimento de bactérias.

Modelos ARMA fazem parte de um conjunto de modelos lineares estacionários. Tais modelos supõem que a série seja gerada por um sistema linear, tendo como entrada ruído branco, como ilustra a Figura 2.4. Pode-se definir $\{\varepsilon_t, t \in Z\}$ como ruído branco discreto, em que Z é um processo estocástico, se as variáveis aleatórias ε_t não são correlacionadas, ou seja, $Cov\{\varepsilon_t, \varepsilon_s\} = 0, t \neq s$, em que Cov é a covariância das variáveis (Walpole et al., 2006). Tal processo é estacionário caso $\forall t, E(\varepsilon_t) = \mu_\varepsilon$ e $Var(\varepsilon_t) = \sigma^2_\varepsilon$, em que $E(\cdot)$ é a esperança da variável aleatória, $Var(\cdot)$ sua variância, μ a média e σ o desvio padrão (Morettin e Toloi, 2006). O sistema, ou filtro, linear pode ser definido de acordo com a Equação 2.3, com saída Z_t, em que a_t é a entrada, $\Psi(B)$ é a função de transferência e B é o operador translação para o passado (definido em 2.2). Esse operador recupera o valor da série em um momento $t - m$, sendo t o instante atual e m o deslocamento que se deseja transladar no tempo.

$$BZ_t = Z_{t-1}, B^m Z_t = Z_{t-m} \tag{2.2}$$

$$Z_t = \mu + a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \ldots = \mu + \psi(B) a_t \tag{2.3}$$

Nesse sistema, μ é um parâmetro que define o nível da série. Caso a sequência dos coeficientes $\psi_j, j \geq 1$ seja finita ou infinita convergente, diz-

se que o filtro é estável, Z_t é estacionária e μ é a média do processo. Definindo $\tilde{Z}_t = Z_t - \mu$, tem-se $\tilde{Z}_t = \psi(B)a_t$. É possível reescrever $\tilde{Z}_t$ como uma soma ponderada de valores passados mais um ruído a_t:

$$\tilde{Z}_t = \pi_1 \tilde{Z}_{t-1} + \pi_2 \tilde{Z}_{t-2} + \ldots + a_t \tag{2.4}$$

Caso $\pi_j = 0, j > p$, obtém-se um modelo denominado auto-regressivo de ordem p, comumente denotado por $AR(p)$. Renomeando os coeficientes π_j para ϕ_j tem-se:

$$\tilde{Z}_t = \phi_1 \tilde{Z}_{t-1} + \phi_2 \tilde{Z}_{t-2} + \ldots + \phi_p \tilde{Z}_{t-p} + a_t \tag{2.5}$$

No caso mais simples, tem-se um modelo auto-regressivo de ordem $p = 1$, ou $AR(1)$, o qual pode ser escrito como:

$$\tilde{Z}_t = \phi \tilde{Z}_{t-1} + a_t \tag{2.6}$$

Nesse caso, Z_t depende somente do valor no instante anterior, Z_{t-1}, e do ruído no instante t.

Modelos ARMA utilizam também conceitos de modelos de médias móveis (MA). Considere novamente o processo linear descrito pela Equação 2.3. Assumindo $\psi_j = 0, j > q$, tem-se um modelo de médias móveis de ordem q, denotado por $MA(q)$. Reescrevendo os coeficientes ψ_j como θ_j, tem-se:

$$Z_t = \mu + a_t + \theta_1 a_{t-1} + \theta_2 a_{t-2} + \ldots + \theta_q a_{t-q} \tag{2.7}$$

Considere $\tilde{Z}_t = Z_t - \mu$. No caso mais simples, obtém-se um modelo de médias móveis de ordem $q = 1$, ou $MA(1)$, que pode ser escrito como:

$$\tilde{Z}_t = a_t + \theta a_{t-1} \tag{2.8}$$

Modelos auto-regressivos de médias móveis, ou ARMA, podem ser definidos como uma combinação de modelos auto-regressivos e de médias móveis, em que p é a ordem da parte auto-regressiva e q é a ordem da parte de médias móveis, denotado por $ARMA(p, q)$:

$$Z_t = \mu + a_t + \phi_1 Z_{t-1} + \ldots + \phi_p Z_{t-p} + \theta_1 a_{t-1} + \ldots + \theta_q a_{t-q} \tag{2.9}$$

A Equação 2.9 pode ser reescrita utilizando a notação de somatório:

$$Z_t = \mu + a_t + \sum_{i=1}^{p} \phi_i Z_{t-i} + \sum_{i=1}^{q} \theta_i a_{t-i} \tag{2.10}$$

Novamente, considere $\tilde{Z}_t = Z_t - \mu$. No caso mais simples, tem-se um modelo ARMA(1,1), que pode ser representado por:

$$\tilde{Z}_t = a_t + \phi Z_{t-1} + \theta a_{t-1} \tag{2.11}$$

Modelos ARMA são adequados para representar séries estacionárias, ou seja, que orbitam em torno de uma média e variância constantes. Entretanto, como já discutido, várias séries reais não se comportam de maneira estacionária, tais como séries de bolsa de valores (Morettin e Toloi, 2006).

Algumas séries não estacionárias apresentam certa semelhança em seu comportamento. Tais séries são denominadas não-estacionárias homogêneas. Esse tipo de série pode ser identificado caso se tome um número finito de diferenças d e ela se torne estacionária. Caso a série representada pela Equação 2.12 seja estacionária, diz-se que é possível transformar a série não estacionária original Z_t na série estacionária W_t. Δ é o operador diferença, o qual computa a diferença entre valores atuais com passados da série, como definido recursivamente na Equação 2.13 e no caso base em 2.14.

$$W_t = \Delta^d Z_t \tag{2.12}$$

$$\Delta^d Z_t = \Delta[\Delta^{d-1} Z_t] \tag{2.13}$$

$$\Delta Z_t = Z_t - Z_{t-1} \tag{2.14}$$

De acordo com Morettin e Toloi (2006), se W_t é uma diferença de Z_t, então Z_t é a integral de W_t, por isso diz-se que Z_t segue um modelo auto-regressivo, integrado, de médias móveis, também chamado de $ARIMA(p, d, q)$. Nesse modelo, p é a ordem do processo auto-regressivo, q é a ordem do processo de médias móveis e d é a ordem do processo integrado, que representa o número de diferenças tomadas para tornar Z_t estacionária. É interessante observar os seguintes casos particulares do modelo ARIMA:

- $ARIMA(p, 0, 0) = AR(p)$;
- $ARIMA(0, 0, q) = MA(q)$;
- $ARIMA(p, 0, q) = ARMA(p, q)$.

Segundo Morettin e Toloi (2006), modelos ARIMA são capazes de representar satisfatoriamente ambas séries estacionárias e não estacionárias, desde que elas não apresentem características explosivas.

Nesse contexto, diversos trabalhos têm empregado modelos ARMA e ARIMA para analisar séries temporais compostas por observações do ambiente em que processos são executados, tais como uso de memória, disco, etc.

Li et al. (2002) estudaram o fenômeno de *software rejuvenation* (Huang et al., 1995), no qual se analisa o ambiente de execução de um processo, a fim de verificar se há exaustão de recursos. A falta de recursos disponíveis pode ser causada por faltas do *software*, tais como vazamentos de memória (*memory leaks*), sistemas de arquivo cheios, *threads* e processos criados e nunca terminados, etc. Nesse contexto, a ideia proposta é reiniciar o processo em execução, a fim de que ele volte a seu estado inicial e esses problemas sejam amenizados. Para tal, busca-se o melhor momento de reinício do processo. Li et al. (2002) propuseram a modelagem dos dados coletados do sistema com modelos ARMA, a fim de detectar e estimar momentos de exaustão de recursos. Os resultados obtidos se mostraram mais eficazes e computacionalmente menos intensivos do que técnicas baseadas em regressão linear.

Zou e Liu (2002) analisaram o problema de congestionamento de redes de computadores de larga escala. Geralmente, administradores de rede só notam problemas de congestionamento após a qualidade dos serviços oferecidos já ter sido afetada. Os autores modelaram o comportamento de tráfego de pacotes *multicast* utilizando um modelo ARMA, a fim de detectar, com antecedência, se o comportamento observado está de acordo com o esperado. Dessa forma, espera-se que sejam tomadas medidas apropriadas de gerenciamento de rede que previnam possíveis falhas.

Sahoo et al. (2003) estudaram a disponibilidade de um aglomerado de computadores no contexto de sistemas *self-healing* (Kephart e Chess, 2003). Esses sistemas visam o auto-gerenciamento e auto-cura, ou seja, devem buscar, ao máximo, automatizar tarefas que previnam suas faltas. Para isso, esses sistemas analisam dados continuamente observados do sistema, a fim de detectar e prever problemas, com o objetivo de tomar atitudes preventivas e evitar falhas catastróficas. Nessa linha, os autores analisaram dados RAS (*Reliability, Availability, Serviceability*) de um aglomerado com 350 computadores. Eles ainda empregaram modelos de séries temporais, tais como AR, MA e ARMA, para modelar o sistema estudado. Variáveis como porcentagem de uso de sistema, tempo ocioso, uso de rede e I/O foram empregadas na construção do modelo, obtendo uma acurácia em torno de 70%.

Falai e Bondavalli (2005) analisaram a qualidade de serviço (QoS) oferecida por diversos detectores de falha em *Wide Area Networks* (WAN's). Entre os testes realizados, foi feita uma análise utilizando modelos de séries temporais, sendo um deles ARIMA. O modelo foi empregado para analisar o comportamento de atrasos de mensagens de *heartbeat* entre dois computadores. Entre os modelos analisados, ARIMA foi o que obteve o menor erro médio quadrado, que representa a média da soma do quadrado da diferença entre valores estimados pelo modelo e os valores reais observados (Hastie et al., 2010).

Esta seção apresentou o emprego de modelos de séries temporais para modelagem do comportamento de ambientes em que processos executam. A fundamentação básica da teoria de modelos auto-regressivos integrados de médias móveis foi apresentada, bem como trabalhos que empregaram modelos desse tipo para analisar sistemas.

Foi possível observar que diversos trabalhos têm empregado séries temporais para modelar sistemas computacionais. Entretanto, modelos como ARMA e ARIMA são lineares, o que nem sempre representa o comportamento de sistemas reais, os quais podem, também, apresentar não linearidade (Ren et al., 2006). Essas limitações levaram ao estudo de outras abordagens, capazes de capturar comportamentos não lineares. Técnicas de aprendizado de máquina, tais como redes neurais artificiais e máquinas de vetores de suporte foram empregadas nesse sentido. Pesquisas realizadas nesse contexto são apresentadas a seguir.

2.3.3 Conceitos e detectores baseados em aprendizado de máquina

Vários trabalhos na área de detecção de faltas de processos têm empregado técnicas de aprendizado de máquina, tais como redes neurais artificiais, máquinas de vetores de suporte, entre outras. Nesta seção, pesquisas que empregam essas abordagens são apresentadas.

Turnbull e Alldrin (2003) analisaram dados de sensores de *hardware* para predizer falhas em um sistema computacional. Os atributos selecionados para monitoramento foram observados de sensores durante janelas de potenciais falhas. Posteriormente, foi realizado o treinamento de uma rede neural de base radial com os dados observados. Os autores concluíram que os dados obtidos de sensores podem auxiliar na predição de falhas de *hardware*, desde que a ocorrência das mesmas não seja frequente.

Ning et al. (2006) estudaram o comportamento de um servidor de aplicação no contexto de envelhecimento de *software*, ou seja, o momento em que há exaustão de recursos como memória, disco, etc. A detecção desse momento permite que a aplicação seja reiniciada antes que os recursos venham a se exaurir. Com esse intuito, foram utilizadas redes *fuzzy wavelet* (FWN) para modelar as séries temporais extraídas de observações de parâmetros de desempenho do servidor. A técnica de análise dos componentes principais (PCA) foi empregada para reduzir a dimensionalidade dos dados. Algoritmos genéticos foram utilizados para encontrar os melhores parâmetros da rede FWN. A técnica proposta pelos autores permitiu detectar a exaustão de recursos antes que o servidor viesse a falhar de fato.

Nesse mesmo contexto de envelhecimento de *software*, El-Shishiny et al.

(2008) investigaram o uso de redes neurais artificiais Multi-Layer Perceptron (MLP), as quais são aproximadores universais de funções contínuas (Hornik et al., 1989). Essa característica permite que essas redes sejam capazes de representar funções complexas, diferentemente de técnicas que consideram simplesmente funções lineares modeladas a partir dos dados. Com esse intuito, os autores utilizaram a rede neural para aprender padrões de uso de memória *swap* e quantidade livre de memória física. Os resultados obtidos com a modelagem não linear, por meio de redes neurais, superaram técnicas clássicas de séries temporais.

Já em um cenário mais voltado para o usuário final, Perkins et al. (2009), do grupo de pesquisa CSAIL (*Computer Science and Artificial Intelligence Laboratory*) do MIT, desenvolveram o sistema *ClearView*, voltado para a correção automática de erros de *software*. O sistema funciona para arquivos binários Windows x86 (não há necessidade de código fonte). O funcionamento básico se dá por meio do aprendizado do comportamento do processo, o que é feito pela identificação de invariantes durante execuções normais. Ao detectar uma execução que viola uma das invariantes encontradas, o sistema altera seu fluxo de execução para tornar tal invariante novamente verdadeira. As invariantes são valores observados em registradores e locais na memória principal. Quanto mais execuções de treinamento, mais o sistema se torna preciso. A versão atualmente implementada utiliza o componente Daikon (Ernst et al., 2007) para realizar esse aprendizado.

Hoffmann et al. (2007) propõem um guia de melhores práticas para construção de modelos empíricos da disponibilidade de sistemas, baseando-se na experiência dos autores na predição de variáveis de desempenho do servidor *Web Apache*. Foram analisadas duas variáveis: tempo de resposta e quantidade de memória física disponível. Após avaliar diversos modelos, os autores concluíram que, entre aqueles estudados, funções de base universal (UBF), propostas pelos autores e baseadas em funções de base radial (RBF), são os melhores preditores para a quantidade de memória disponível do servidor Apache. Entre os modelos analisados, máquinas de vetores de suporte (SVM) apresentaram a maior eficácia tanto para predições de curto prazo, quanto de longo prazo, do tempo de resposta do servidor.

Segundo Lorena e Carvalho (2007), máquinas de vetores de suporte (SVM) compõem uma técnica de aprendizado de máquina que tem sido utilizada em diversos cenários, tais como categorização de textos, análise de imagens e Bioinformática. De acordo com os autores, as SVMs têm apresentado, muitas vezes, resultados melhores do que outras técnicas disponíveis, como redes neurais artificiais.

As SVMs surgiram a partir da Teoria de Aprendizado Estatístico (TAE), de-

senvolvida por Vapnik (Vapnik, 2000). Esse aprendizado prescreve conceitos sobre como é possível obter classificadores com bom nível de generalização, ou seja, classificadores capazes de discriminar corretamente exemplos não vistos durante a etapa de treinamento.

Considere o caso de aprendizado de máquina supervisionado, em que dado um conjunto de exemplos X, tem-se um conjunto de valores esperados Y. A matriz X é composta por n exemplos de treinamento $x_i = (x_{i1}, x_{i_2}, \ldots, x_{im})$, sendo m o número de atributos de cada exemplo. O vetor Y representa as respostas esperadas, cujos valores podem ser discretos ou contínuos.

O objetivo de uma técnica de aprendizado de máquina é induzir um classificador $f \in F$. O conjunto F representa todos os classificadores que podem ser gerados pela técnica de aprendizado. A indução ocorre a partir de um conjunto de treinamento T, que consiste de n exemplos do tipo (x_i, y_i). Pode-se entender f, também, como uma função que mapeia padrões de entrada x_i para saídas $f(x_i)$.

De acordo com Muller et al. (2001), a TAE assume que os dados utilizados para o aprendizado foram gerados de maneira i.i.d., ou seja, independente e identicamente distribuídos, seguindo uma distribuição de probabilidade $P(x, y)$. Isso permite definir o risco esperado, também chamado de erro verdadeiro, sobre dados de validação de um classificador treinado no conjunto de todos os possíveis padrões (Passerini, 2004):

$$R(f) = \int c(f(x), y) dP(x, y) \tag{2.15}$$

Na Equação 2.15, $c(f(x), y)$ é uma função custo que atribui pesos a exemplos classificados incorretamente, enquanto $dP(x, y)$ é a função densidade de probabilidade de x, y. Para problemas de classificação binários (com duas classes), em que $y \in \{-1, 1\}$, a função de custo pode ser dada por 2.16, também chamada de função custo $0/1$, pois retorna custo 0 no caso de uma classificação correta e 1 caso incorreta. Para problemas de regressão ($y \in \mathbb{R}$), a função de custo pode ser dada por 2.17.

$$c(f(x), y) = \theta(-yf(x)), \theta(z) = \begin{cases} 0, z < 0 \\ 1, z \geq 0 \end{cases} \tag{2.16}$$

$$c(f(x), y) = (f(x) - y)^2 \tag{2.17}$$

O risco esperado pode ser utilizado como uma medida da capacidade de generalização do classificador f. Entretanto, não há como minimizar $R(f)$ diretamente, dado que $P(x, y)$ não é conhecida (Passerini, 2004). Geralmente, busca-se minimizar o erro do classificador sobre os dados de treinamento, já que esse pode de fato ser medido, e espera-se que isso também minimize o

erro sobre os dados de teste.

De acordo com Passerini (2004), chama-se de risco empírico R_{emp} de um classificador f, a taxa média de erro sobre dados do conjunto de treinamento T, definido pela Equação 2.18.

$$R_{emp} = \frac{1}{n} \sum_{i=1}^{n} c(f(x_i), y_i) \tag{2.18}$$

Por indução, a minimização do risco empírico, quando o número de exemplos $n \to \infty$, possibilita que ocorra a convergência do risco empírico para o risco esperado (Muller et al., 2001). Entretanto, a priori não existem garantias de que um classificador com risco empírico zero tenha, também, risco esperado zero. Esse é o caso de um classificador que memoriza todos os exemplos de treinamento e posteriormente realiza classificações aleatórias (Passerini, 2004). Para generalizar f para padrões não vistos durante o treinamento, é necessário restringir o conjunto de funções que podem ser induzidas de F, levando em conta a capacidade ou complexidade dessas funções, dada uma determinada tarefa de aprendizado e o número de exemplos de treinamento disponíveis.

Segundo Lorena e Carvalho (2007), a escolha arbitrária de um classificador com baixo risco empírico pode ser perigosa, pois pode ocorrer o que é chamado de super-ajustamento ao conjunto de treinamento, ou *overfitting*. Nesse caso, o classificador se especializa em exemplos em T, porém não consegue generalizar seu aprendizado para outras instâncias. A TAE fornece um meio pelo qual é possível limitar o risco esperado de um classificador, levando em conta a complexidade (capacidade) do mesmo, o que auxilia na escolha de $f \in F$.

Para definir os limites estabelecidos pela TAE ao conjunto de funções induzíveis, é necessário primeiro definir o conceito de dimensão VC, ou dimensão Vapnik-Chervonenkis (Vapnik e Chervonenkis, 1971). Essa é uma medida da complexidade do conjunto de classificadores possíveis de serem gerados em F. Valores maiores da dimensão VC indicam que o nível de complexidade das funções possíveis de serem induzidas é maior.

De acordo com Passerini (2004), dado um problema binário de classificação, a dimensão VC pode ser entendida como o número máximo n de exemplos x_i divisíveis nas duas classes $\{-1, 1\}$ pelos classificadores que podem ser induzidos em F, levando em conta todas as possíveis combinações 2^n dos dados. Se o máximo não existir, diz-se que a dimensão VC é infinita. Burges (1998) provou que a dimensão VC de um conjunto de hiperplanos orientados em $\mathbb{R}^n$ é $n + 1$.

É interessante estudar essa questão no contexto da indução de um classificador que resolve o problema do ou-exclusivo (*xor*). Considere o plano $\mathbb{R}^2$, e F o conjunto das retas. Não é possível traçar uma reta para separar os qua-

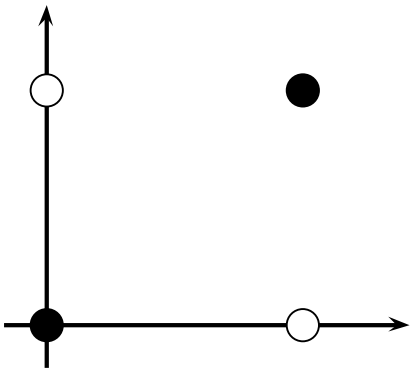


Figura 2.5: Representação de pontos da função ou exclusivo. Não é possível traçar uma reta no plano $\mathbb{R}^2$, de forma que os pontos da mesma classe fiquem separados dos pontos da outra classe. Trata-se de um problema não linearmente separável.

tro pontos apresentados na Figura 2.5. Entretanto, sempre é possível separar três pontos em $\mathbb{R}^2$ utilizando uma reta, logo, tem-se que a dimensão VC é 3.

Segundo Burges (1998), um dos limites que a Teoria de Aprendizado Estatístico enuncia é o definido pela Inequação 2.19, que fornece um limitante para o risco esperado, com base no risco empírico do classificador e um termo de capacidade, baseado na dimensão VC.

$$R(f) \leq R_{emp}(f) + \sqrt{\frac{h(\ln(2n/h) + 1) - \ln(\theta/4)}{n}} \tag{2.19}$$

Na Inequação 2.19, denomina-se de capacidade o termo na raiz somado ao risco empírico do classificador (Lorena e Carvalho, 2007). O h expressa a dimensão VC de F e n é o número de exemplos no conjunto de treinamento T. Esse limite é garantido com uma probabilidade $1 - \theta$, sendo $\theta \in [0, 1]$.

De acordo com Lorena e Carvalho (2007), o limite apresentado na Inequação 2.19 é importante, pois fornece um meio para controlar a capacidade de F, de onde o classificador será induzido. Ele permite minimizar o risco esperado por meio da escolha de um f que minimize o risco empírico e que esteja em um conjunto F de baixa dimensão VC.

O gráfico apresentado na Figura 2.6 ilustra o comportamento do termo de capacidade para diferentes valores da dimensão VC, assumindo $\theta = 0,1$ e um conjunto de dados de tamanho $n = 1000$. Observa-se que, quanto maior a dimensão VC, ou seja, maior a complexidade das funções induzíveis em F, maior também é o termo de capacidade, somado no cálculo de risco na Inequação 2.19.

Segundo Vapnik (2000), o limite apresentado define um princípio da TAE denominado minimização do risco estrutural. Essa técnica busca dividir o conjunto de funções F em subconjuntos $F_i \subset F, i = 1, 2, 3, \ldots$, tal que $F_1 \subset F_2 \subset \ldots \subset F$ e a capacidade h (dimensão VC) aumenta conforme i cresce, $h_1 < h_2 < \ldots < h$. Denomina-se cada $F_i \subset F$ uma estrutura.

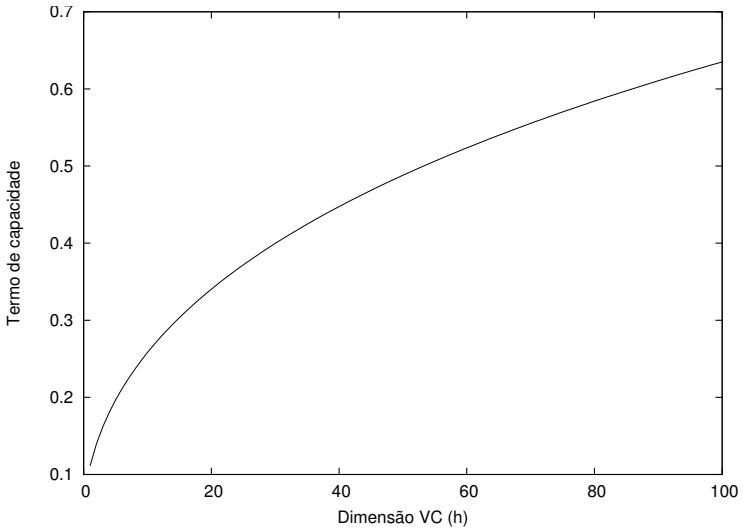


Figura 2.6: Gráfico do termo de capacidade variando o valor da dimensão VC (h), assumindo como fixo ($n = 1000$) o tamanho do conjunto de treinamento T e $\theta = 0,1$, que faz a Inequação 2.19 ser garantida com probabilidade = 90%.

De acordo com Lorena e Carvalho (2007), considerando um subconjunto F_k de classificadores em F e f_k o classificador com menor risco empírico, tem-se que quanto maior k, menor será o risco empírico $R_{emp}(f_k)$, já que é possível representar funções mais complexas. Entretanto, a dimensão VC (h), e consequentemente o termo de capacidade, aumentam. Busca-se, portanto, o classificador que tenha o melhor equilíbrio entre risco empírico e capacidade, de forma que sua soma seja minimizada, o que consequentemente minimiza o risco esperado $R(f)$. Essa troca entre capacidade e risco empírico estabelece o fundamento da minimização do risco estrutural.

Segundo Lorena e Carvalho (2007), o limite definido pela Inequação 2.19 é útil para compreender os princípios da minimização do risco estrutural. Entretanto, na prática, a dimensão VC nem sempre é trivialmente computada, pois pode ser infinita ou nem mesmo existir. Por outro lado, para funções de decisão que são lineares, do tipo $f(x) = w \cdot x$, pode-se relacionar o risco esperado ao conceito de margem.

De acordo com Smola (2000), a margem de um classificador tem relação com a confiança com que uma classificação é realizada, o que pode ser computado pela distância de um exemplo à fronteira de decisão induzida. Para problemas binários ($y \in \{-1, 1\}$), a margem $\varrho(f(x_i), y_i)$ com que um exemplo x_i é classificado, é dada por $y_i f(x_i)$, o que resulta em uma margem negativa no caso de classificação errônea (Lorena e Carvalho, 2007). Esse conceito permite definir o erro marginal de uma função $f(R_\rho(f))$ sobre dados de treinamento T, que computa a proporção de exemplos de treinamento classificados com uma margem de confiança menor do que uma constante $\rho > 0$:

$$R_\rho(f) = \frac{1}{n}\sum_{i=1}^{n} I(y_i f(x_i) < \rho); \begin{cases} I(z) = 1, \text{ se } z \text{ verdadeiro} \\ I(z) = 0, \text{ caso contrário} \end{cases} \tag{2.20}$$

Segundo Smola (2000), dada uma esfera de raio R em $\mathbb{R}^n$, centrada na origem, existe uma constante c, que com probabilidade $1-\theta$ ($\theta \in [0,1]$), $\forall \rho > 0$, e F sendo o conjunto de funções lineares da forma $f(x) = w \cdot x$, com $||x|| \leq R$ e $||w|| \leq 1$, aplica-se o limite:

$$R(f) \leq R_\rho(f) + \sqrt{\frac{c}{n}\left(\frac{R^2}{\rho^2}\log^2\left(\frac{n}{\rho}\right) + \log\left(\frac{1}{\theta}\right)\right)} \tag{2.21}$$

Segundo Lorena e Carvalho (2007), esse limite define que, dada uma maior margem ρ, tem-se um menor termo de capacidade, o que reduz o risco. Porém, aumentar a margem pode também aumentar o erro marginal ($R_\rho(f)$), pois é mais difícil garantir que todos os exemplos x_i estejam distantes de uma margem ρ em relação ao hiperplano divisor. Já um valor baixo para a margem leva a um erro marginal menor sobre os dados de treinamento, mas maximiza o termo de capacidade. É necessário, portanto, encontrar um balanço entre uma margem maior e um erro marginal menor. Um hiperplano é denominado ótimo se sua margem ρ é máxima enquanto o erro marginal é mínimo, de forma que se minimiza o erro em ambos os dados de treinamento e de teste.

Considere o problema de classificação binário em que $y \in \{-1, 1\}$. De acordo com Lorena e Carvalho (2007), SVMs lineares separam os dados utilizando hiperplanos, os quais possuem a forma da Equação 2.22, na qual w é o vetor normal ao hiperplano, x representa um vetor de exemplo x_i, e $\frac{b}{||w||}$ é a distância do hiperplano à origem, em que $b \in \mathbb{R}$. Esse hiperplano realiza a divisão dos dados X em duas regiões, as quais são utilizadas para determinar a classe de um exemplo, como apresentado na Equação 2.23.

$$f(x) = w \cdot x + b \tag{2.22}$$

$$\varrho(x) = sinal(f(x)) = \begin{cases} +1, w \cdot x + b > 0 \\ -1, w \cdot x + b < 0 \end{cases} \tag{2.23}$$

Tomando $f(x)$, pode-se obter hiperplanos equivalentes pela multiplicação de w e b por uma constante. Denomina-se de hiperplano canônico com respeito ao conjunto de treinamento T, o hiperplano em que w e b atendam à restrição de que os exemplos mais próximos dele satisfaçam (Lorena e Carvalho, 2007):

$$|w \cdot x_i + b| = 1 \tag{2.24}$$

Tal restrição implica na inequação:

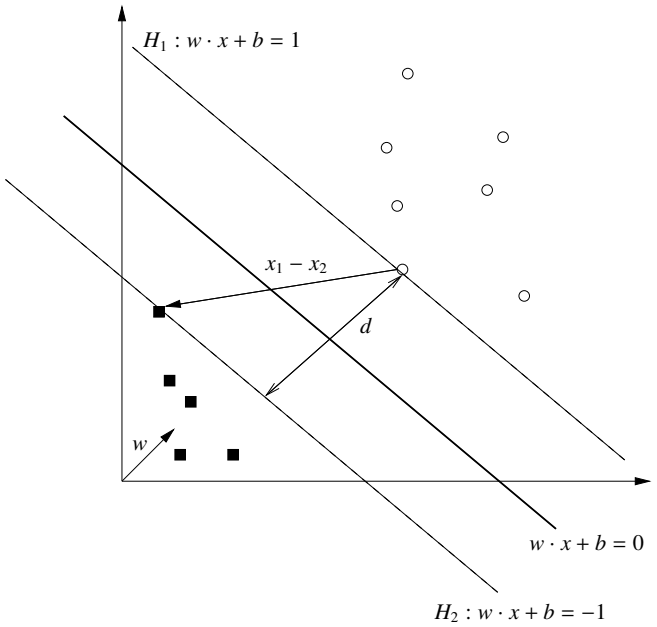


Figura 2.7: Hiperplano separador $w \cdot x + b = 0$, juntamente com a distância d entre as margens definidas por H_1 e H_2 (Lorena e Carvalho, 2007).

$$y_i(w \cdot x_i + b) - 1 \geq 0, \forall (x_i, y_i) \in T \tag{2.25}$$

De acordo com Lorena e Carvalho (2007), tomando x_1 como um ponto no hiperplano $H_1 : w \cdot x_i + b = 1$ e x_2 em $H_2 : w \cdot x_i + b = -1$, tem-se que projetando $x_1 - x_2$ na direção de w, obtém-se a distância entre H_1 e H_2, o que é ilustrado na Figura 2.7. A projeção é dada por:

$$(x_1 - x_2)\left(\frac{w}{||w||} \cdot \frac{(x_1 - x_2)}{||x_1 - x_2||}\right) \tag{2.26}$$

A diferença entre as equações de H_1 e H_2 fornecem $w \cdot (x_1 - x_2) = 2$. Substituindo em 2.26 obtém-se:

$$\frac{2(x_1 - x_2)}{||w||||x_1 - x_2||} \tag{2.27}$$

Para obter o comprimento do vetor projetado, computa-se a norma de 2.27:

$$d = \frac{2}{||w||} \tag{2.28}$$

O valor d obtido é a distância entre os hiperplanos H_1 e H_2, ambos paralelos à fronteira de decisão induzida (Lorena e Carvalho, 2007).

De acordo com Smola (2000), em função de w e b terem sido escalados para não haver exemplos entre H_1 e H_2, a maximização da margem de separação ρ dos dados com relação à fronteira de decisão induzida, é dada pela minimização de $||w||$, que é um problema de otimização:

$$\text{Minimizar}_{w,b} : \frac{1}{2}||w||^2 \tag{2.29}$$

atendendo a:

$$y_i(w \cdot x_i + b) - 1 \geq 0, \forall i = 1, \ldots, n \tag{2.30}$$

As restrições impostas garantem que não existem exemplos entre as margens de separação (Lorena e Carvalho, 2007). A resolução se reduz a um problema de otimização quadrático. Como a função minimizada é convexa, tem-se, também, um mínimo global. Tais tipos de problema possuem solução matemática conhecida (Smola, 2000). O nome da técnica, vetores de suporte, reside no fato de que esses vetores são os dados mais informativos do conjunto de treinamento T, pois somente eles participam na determinação do hiperplano separador, que decorre da solução do problema de otimização quadrático.

A limitação de muitas das técnicas de aprendizado de máquina, quando aplicadas ao problema de tolerância a faltas, é que várias delas assumem os dados como i.i.d., ou seja, que foram retirados de amostras independentes e que seguem uma mesma distribuição de probabilidade. Entretanto, séries de comportamento de sistema apresentam relações temporais entre si. Por exemplo, uma falta no passado pode afetar o comportamento atual do sistema, degradando tempo de resposta, utilizando mais memória, etc. Esse tipo de relação não é detectada quando se assume i.i.d.

2.4 Considerações finais

Neste capítulo apresentou-se uma revisão de trabalhos relacionados com técnicas para detecção de faltas e falhas. Entre as abordagens existentes, observou-se uma divisão em três grupos proeminentes: abordagens baseadas em *hearbeat*, estatísticas e de aprendizado de máquina. Técnicas baseadas em *heartbeats* detectam falhas, enquanto as baseadas em estatística e aprendizado de máquina podem, de fato, detectar faltas. Apresentou-se com maiores detalhes a técnica estatística ARIMA, que cria modelos auto regressivos integrados de médias móveis, bem como a técnica de aprendizado de máquina SVM, que busca pelo melhor hiperplano de separação dos dados. No capítulo seguinte, discute-se conceitos relacionados à abordagem proposta para detecção de faltas.

CAPÍTULO

3

Conceitos relacionados

3.1 Considerações iniciais

Este capítulo discute conceitos utilizados na proposta da nova abordagem para detecção de faltas. Apresenta-se, inicialmente, redes neurais artificiais com funções de base radial (RBF), as quais estão relacionadas à identificação de estados representativos de um processo. Posteriormente introduz-se cadeias de Markov, as quais são empregadas para modelar as transições entre os estados identificados. Finalmente, conceitos de Teoria da Informação e Entropia são discutidos, os quais são utilizados para computar níveis de novidade sobre cadeias de Markov estimadas.

3.2 Redes neurais artificiais RBF – Funções de base radial

De acordo com Haykin (2008), enquanto redes neurais *Multi-Layer Perceptrons* realizam seu aprendizado utilizando uma técnica de aproximação estocástica, outras redes neurais, como *Radial Basis Function Neural Networks* (RBFNNs), empregam uma técnica de aproximação de função em um espaço multidimensional. O problema de aprendizado é reduzido a encontrar uma superfície em um espaço multidimensional, sendo que essa superfície forneça o melhor ajuste (*fit*) aos dados de treinamento. Esse ajuste pode ser computado por meio de medidas de erro, tais como o erro quadrado e o erro absoluto.

Segundo Haykin (2008), a topologia de uma rede neural RBF consiste de três camadas, conforme ilustrado na Figura 3.1. A camada de entrada é com-

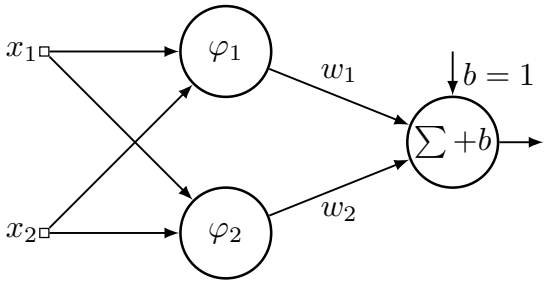


Figura 3.1: Exemplo de uma rede RBF com dois neurônios na camada escondida, representados pelas funções de base radial φ_1 e φ_2. O somatório $\sum$ representa a soma dos produtos dos pesos pela ativação de cada função de base radial. (Haykin, 2008).

posta de neurônios de sensoriamento, que conectam a rede a seu ambiente. A segunda camada é escondida e aplica uma transformação não linear do espaço de entrada para o espaço escondido, com alta dimensão para a maioria das aplicações. A camada de saída é linear e fornece a saída da rede para o padrão de entrada recebido. Na Figura 3.1, os termos x_1 e x_2 representam atributos de um padrão de entrada x, enquanto φ_1 e φ_2 são funções de base radial associadas aos neurônios. Os pesos da rede são representados por w_1 e w_2, enquanto o *bias* por b.

Cover (1965) enuncia um teorema sobre a separabilidade de padrões, o qual afirma que a probabilidade de um problema de classificação ser linearmente separável aumenta quando esse é levado para um espaço de alta dimensão. Essa é a razão pela qual a camada escondida de uma rede RBF apresenta, geralmente, alta dimensão. Além disso, quanto maior a dimensão do espaço escondido, mais precisa é a aproximação da função realizada (Mhaskar, 1996).

Haykin (2008) observa, entretanto, que a transformação não linear, em alguns casos, pode ser suficiente para tornar os padrões linearmente separáveis, sem a necessidade de aumentar a dimensão dos dados. Considere o problema do aprendizado da função ou-exclusivo (*xor*). Nesse problema, deve-se aprender a discriminar quatro padrões de entrada – $(0,0), (0,1), (1,0), (1,1)$ – em um espaço bidimensional, entre duas classes diferentes. Sejam duas funções Gaussianas definidas como:

$$\varphi_1(x) = \exp^{-||x-t_1||^2}, t_1 = [1,1] \tag{3.1}$$

$$\varphi_2(x) = \exp^{-||x-t_2||^2}, t_2 = [0,0] \tag{3.2}$$

Aplicando as funções escondidas 3.1 e 3.2 aos padrões de entrada, tem-se como resultado as coordenadas da Tabela 3.1.

Ao mapear os padrões de entrada para o plano $\varphi_1 - \varphi_2$, o problema torna-se linearmente separável, como pode ser visto na Figura 3.2. Dessa forma, mesmo sem aumentar a dimensão do espaço em que o problema está inse-

Tabela 3.1: Transformação não linear, que torna o problema ou-exclusivo em um problema linearmente separável.

Padrão de Entrada (x)	**Função** $\varphi_1(x)$	**Função** $\varphi_2(x)$
[1,1]	1	0,1353
[0,1]	0,3678	0,3678
[1,0]	0,3678	0,3678
[0,0]	0,1353	1

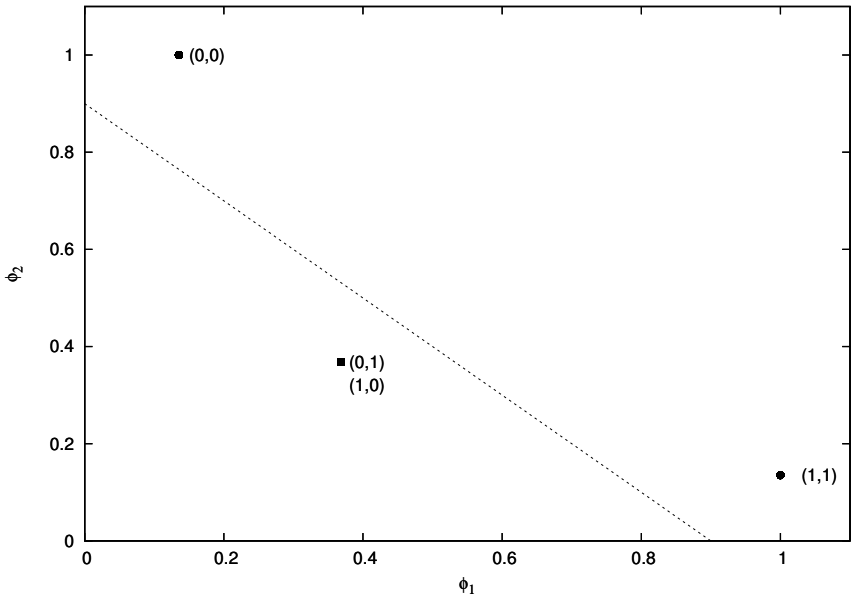


Figura 3.2: Uma transformação não linear pode tornar a função ou-exclusivo em um problema linearmente separável, simplificando sua solução (Haykin, 2008).

rido (continuou-se a trabalhar no plano $\mathbb{R}^2$), a transformação não linear realizada pelas funções Gaussianas foi suficiente para tornar o problema do ou-exclusivo em linearmente separável, simplificando sua solução.

De acordo com Powell (1988), a técnica RBF consiste na escolha de uma função F da seguinte forma:

$$F(x) = \sum_{i=1}^{N} w_i \varphi(||x - x_i||) \tag{3.3}$$

Na Equação 3.3, $\{\varphi(||x - x_i||), i = 1, 2, \ldots, N\}$ é um conjunto de N funções não lineares, também chamadas de funções de base radial. $||\cdot||$ representa uma norma, geralmente tomada como a distância Euclidiana. O vetor w_i representa pesos atribuídos às saídas de cada uma das funções. Os pontos $x_i, i = 1, 2, \ldots, N$ são as coordenadas dos centros das funções de base radial.

De acordo com Davis (1963), o problema de generalização da rede pode ser enunciado da seguinte forma: dados N pontos diferentes no espaço de dimensão m_0, $\{x_i \in \mathbb{R}^{m_0} | i = 1, 2, \ldots, N\}$, e um conjunto correspondente de N números reais, $\{d_i \in \mathbb{R} | i = 1, 2, \ldots, N\}$, encontre uma função $F : \mathbb{R}^{m_0} \to \mathbb{R}$ que satisfaça a condição:

$$F(x_i) = d_i, i = 1, 2, \ldots, N \tag{3.4}$$

Aplicando as restrições da Equação 3.4 em 3.3, tem-se um sistema linear, no qual se deseja descobrir o vetor w_i de pesos (Haykin, 2008):

$$\begin{matrix} \Phi & & & & \cdot & w & = & d \end{matrix}$$
$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{N1} & \varphi_{N2} & \cdots & \varphi_{NN} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix} \tag{3.5}$$

Na Equação 3.5, $\varphi_{ji} = \varphi(||x_j - x_i||), (j, i) = 1, 2, \ldots, N$. Seja:

$$d = [d_1, d_2, \ldots, d_N]^T \tag{3.6}$$

$$w = [w_1, w_2, \ldots, w_N]^T \tag{3.7}$$

O vetor d denota o conjunto de respostas desejadas, enquanto w é o vetor de pesos, em que N é o tamanho do conjunto de treinamento. A matriz Φ, de ordem $N \times N$, com elementos φ_{ji}, é denominada matriz de interpolação. O sistema linear apresentado na Equação 3.5 pode ser escrito de forma compacta:

$$\Phi w = d \tag{3.8}$$

Dessa forma, é possível encontrar o vetor de pesos adequado para a rede RBF de forma fechada. Para isso, deve-se multiplicar a Equação 3.8 pela inversa Φ^{-1}.

$$w = \Phi^{-1} d \tag{3.9}$$

Entretanto, para que a solução apresentada seja viável, é necessário que a matriz de interpolação Φ seja não singular, ou seja, que apresente determinante diferente de zero, caso contrário a inversa Φ^{-1} não existirá. Micchelli (1986) provou que uma vasta classe de funções de base radial apresenta uma matriz de interpolação não singular. Entre outras, nesse conjunto de funções estão:

- Multiquádricas: $\varphi(r) = (r^2 + c^2)^{1/2}, c > 0, r \in \mathbb{R}$;
- Multiquádricas inversas: $\varphi(r) = \frac{1}{(r^2+c^2)^{1/2}}, c > 0, r \in \mathbb{R}$;
- Gaussianas: $\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right), \sigma > 0, r \in \mathbb{R}$.

Micchelli (1986) mostrou que, para que essas funções tenham uma matriz de interpolação Φ não singular, basta que todos os pontos $x_{i i=1}^{N} \in \mathbb{R}^{m_0}$ sejam distintos, independente do tamanho N do conjunto de treinamento e da dimensão m_0 dos vetores x_i.

Bishop (2007) argumenta, por outro lado, que o método exato para encontrar os pesos de uma rede RBF, como apresentado anteriormente, pode não ser o ideal para tarefas de reconhecimento de padrões. Isso deve-se ao fato de que ao utilizar a equação para encontrar os pesos de forma fechada, pode-se obter uma solução sobre-ajustada (*overfitted*) aos dados de treinamento. Bishop (2007) propõe que outros métodos de treinamento sejam utilizados, como o ajuste dos pesos com base em uma função de erro. Uma possível função para essa tarefa é a que mede o erro quadrático, conforme mostra a Equação 3.10.

$$E = \frac{1}{2}(y(x_n) - d_n)^2 \tag{3.10}$$

Na Equação 3.10, $y(x_n)$ é a saída da rede para a entrada x_n e d_n é a saída desejada para essa entrada.

Neste trabalho, funções de base radial (RBFs) são utilizadas para agrupar dados. Para cada padrão de entrada computam-se as distâncias dele até cada centro da rede. Cada distância é utilizada como entrada para a função de ativação de cada centro. O padrão é agrupado ao centro que obtiver maior ativação. Dessa forma, busca-se encontrar uma correspondência entre estados representativos de um processo com centros da rede, conforme será discutido no Capítulo 4. Após a etapa de agrupamento, é estimada uma cadeia de Markov que representa o relacionamento entre diferentes estados.

3.3 Cadeias de Markov

Segundo Ross (2003), um processo estocástico $\{X_t, t \in T\}$ é uma coleção de variáveis aleatórias, ou seja, $\forall t \in T, X_t$ é uma variável aleatória. Assume-se geralmente o índice t como tempo, e X_t como o estado do processo no instante t.

Considere um processo estocástico $\{X_n, n = 0, 1, 2, \ldots\}$ que assume um número contável de possíveis valores. Se $X_n = i$, diz-se que o processo está no estado i no tempo n (Ross, 2003). Sempre que o processo está no estado i existe uma probabilidade fixa P_{ij} de que ele irá para o estado j no próximo instante. Diz-se que:

$$P\{X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots, X_1 = i_1, X_0 = i_0\} = P_{ij} \tag{3.11}$$

para todos os estados $i_0, i_1, \ldots, i_{n-1}, i, j$ e todo $n \geq 0$. Um processo estocástico dessa forma é chamado de Cadeia de Markov (Ross, 2003). A Equação 3.11 exprime a probabilidade P_{ij} do processo ir do estado i para o estado j. Como as probabilidades são não negativas e o processo deve ir de um estado para outro, tem-se:

$$P_{ij} \geq 0, i, j \geq 0 \tag{3.12}$$

$$\sum_{j=0}^{\infty} P_{ij} = 1, i = 0, 1, \ldots \tag{3.13}$$

Com isso, é possível definir a matriz P de transições de 1 passo. Seus elementos P_{ij} representam a probabilidade de transição do estado i para o estado j e $0, 1, \ldots, N$ são os possíveis estados:

$$P = \begin{bmatrix} P_{00} & P_{01} & \cdots & P_{0N} \\ P_{21} & P_{22} & \cdots & P_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ P_{N1} & P_{N2} & \cdots & P_{NN} \end{bmatrix} \tag{3.14}$$

É possível definir, também, uma matriz de n-passos P_{ij}^n que diz a probabilidade do processo ir para o estado j partindo do estado i após n transições adicionais (Ross, 2003):

$$P_{ij}^n = P\{X_{k+n} = j | X_k = i\}, n \geq 0, i, j \geq 0 \tag{3.15}$$

A equação de Chapman-Kolmogorov fornece um método para computar essas probabilidades após n-passos:

$$P_{ij}^{n+m} = \sum_{k=0}^{\infty} P_{ik}^n P_{kj}^m, \forall n, m \geq 0 \tag{3.16}$$

Segundo Ross (2003), na Equação 3.16, $P_{ik}^n P_{kj}^m$ representa a probabilidade de que, iniciando no estado i, o processo irá para o estado j em $n + m$ transições, por meio de um caminho que o leva ao estado k na n-ésima transição.

Assim, a soma a probabilidade de transição de todos os estados intermediários k fornece a probabilidade de que o processo estará no estado j após $n + m$ transições. Denotando por $P^{(n)}$ a matriz de n-passos, tem-se:

$$P^{(n+m)} = P^{(n)} \cdot P^{(m)} \tag{3.17}$$

Para uma matriz de 2-transições tem-se:

$$P^{(2)} = P \cdot P = P^2 \tag{3.18}$$

Por indução:

$$P^{(n)} = P^{(n-1+1)} = P^{n-1} \cdot P = P^n \tag{3.19}$$

Dessa forma, é possível obter a matriz de n-transições multiplicando a matriz P por si mesma n vezes. No limite em que $n \to \infty$, tem-se uma matriz que representa a probabilidade do processo encontrar-se em cada um dos estados para qualquer instante. Essa matriz representa o *steady-state* do processo (Ching et al., 2007).

Neste trabalho, cadeias de Markov são utilizadas para representar relações entre diferentes estados de um processo. Tais relações são utilizadas para capturar a dinâmica do comportamento normal de um processo. Detecta-se que o comportamento divergiu do esperado, ou seja, que uma falta ocorreu, quando há uma variação significativa nas probabilidades estimadas para a cadeia de transições de um processo.

3.4 Teoria da Informação

De acordo com Freeman e Skapura (1991), na Teoria da Informação tem-se que um evento e pode ocorrer com probabilidade $P(e)$. Caso esse evento venha a ocorrer de fato, então se tem:

$$I(e) = \log_2 \frac{1}{P(e)} \tag{3.20}$$

A Equação 3.20 define o número de *bits* (*binary digits*) recebidos de informação quando e ocorreu, conforme postula a Teoria da Informação (Freeman e Skapura, 1991). Considere o lançamento de uma moeda honesta, em que $e = coroa$. A probabilidade $P(e) = 50\%$, nesse caso. Tem-se então que $I(e) = 1$ *bit*.

Segundo Freeman e Skapura (1991), pode-se definir um *bit* como a quantidade de informação recebida quando uma, entre duas alternativas possíveis, ocorre. Caso se saiba com certeza que um evento ocorre, ele não fornece nenhuma nova informação, o que é representado por $I(e) = \log_2(1/1)$, que resulta em 0. A maior quantidade de informação é recebida quando menos certeza se tem sobre a ocorrência de um determinado evento. O gráfico da Figura 3.3 ilustra essa relação.

De acordo com Freeman e Skapura (1991), pode-se definir uma fonte com zero-memória como uma fonte de informação que tem como saída um conjunto de símbolos $S = \{s_1, s_2, \ldots, s_q\}$, com cada símbolo ocorrendo com uma probabilidade fixa $\{P(s_1), P(s_2), \ldots, P(s_q)\}$, sendo que a probabilidade de um símbolo independe da probabilidade de outro ter ocorrido. Fontes de zero-

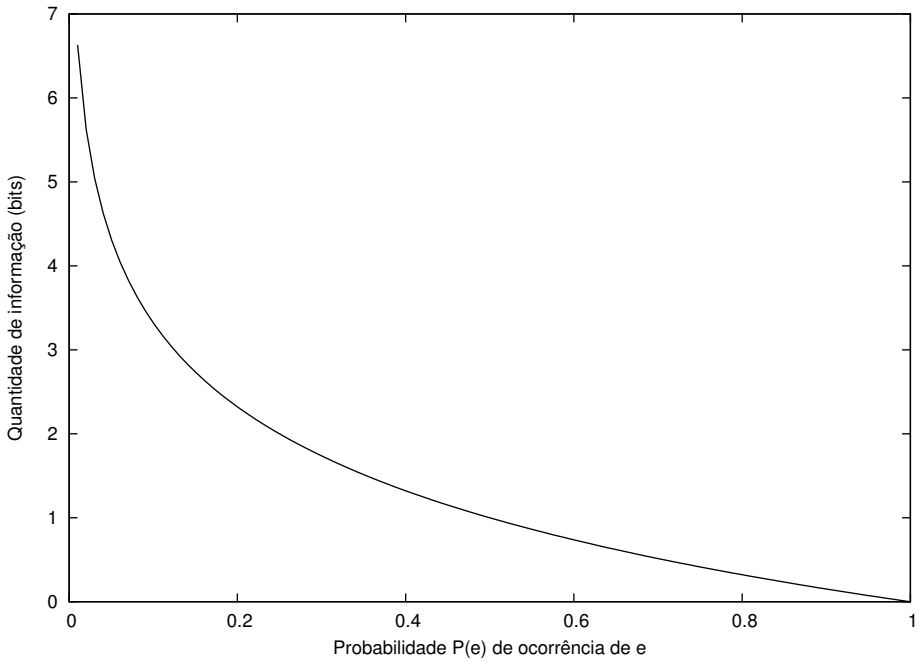


Figura 3.3: O número de *bits*, ou quantidade de informação fornecida, por um evento e que ocorre com probabilidade $P(e)$. Quanto maior a certeza sobre um evento, menor é a quantidade de informação recebida quando ele ocorre.

memória apresentam o seguinte nível de informação a cada símbolo recebido:

$$I(s_i) = \log_2 \frac{1}{P(s_i)} \text{ para } i = 1, \ldots, q. \tag{3.21}$$

O valor médio de informação recebida por símbolo pode ser computado como a esperança de I:

$$E(I) = \sum_{i=1}^{q} P(s_i) I(s_i) \tag{3.22}$$

Substituindo 3.21 em 3.22, tem-se:

$$E(I) = -\sum_{i=1}^{q} P(s_i) \log_2 \left(P(s_i)\right) \tag{3.23}$$

A Equação 3.23 calcula a Entropia proposta por Shannon (2001). De forma geral, pode-se entender a Entropia como uma medida de impureza, ou desordem. Em um sistema no qual todos os eventos ocorrem com mesma probabilidade, tem-se o maior nível de Entropia, ou incerteza.

Neste trabalho, a Entropia é utilizada como uma medida de novidade associada a um processo, o qual é modelado por uma cadeia de Markov. As mudanças nas probabilidades da cadeia, ao longo do tempo, são utilizadas como entrada para o cálculo da Entropia, que fornece o nível de novidade para o processo. Esse nível indica, caso acima de um limiar estabelecido, se o processo divergiu de seu padrão esperado, de forma que se considera que uma falta ocorreu.

3.5 Considerações finais

Este capítulo apresentou conceitos subjacentes à abordagem proposta neste trabalho. Conceitos de funções de base radial (RBF) são utilizados para encontrar estados representativos de um processo, enquanto cadeias de Markov e Entropia fornecem meios para medir níveis de novidade, a qual é utilizada no contexto deste trabalho para indicar faltas. No capítulo seguinte a abordagem proposta é detalhada.

CAPÍTULO

4

Abordagem Proposta

4.1 Considerações iniciais

Neste capítulo, apresenta-se a abordagem proposta para detecção de faltas, discutindo o seu funcionamento bem como sua complexidade assintótica.

4.2 Abordagem Proposta

A abordagem proposta considera funções de base radial (Haykin, 2008), cadeias de Markov (Ross, 1995) e Entropia (Shannon, 2001), para detectar novidades no comportamento de processos. Nesse contexto, novidades podem corresponder a estados errôneos visitados, os quais são diferentes do comportamento normal esperado.

O primeiro passo da abordagem é computar a distância Euclidiana de cada padrão de entrada aos centróides[1] já criados, os quais representam informações conhecidas, ou o comportamento normal modelado. A distância é calculada utilizando a Equação 4.1, que considera o padrão de entrada (I) e os centróides (C). Na equação, I_{ik} e C_{jk} são atributos do padrão de entrada i e do centróide j, respectivamente, e n é o número de atributos.

$$D(I_i, C_j) = \sqrt{\sum_{k=1}^{n} (I_{ik} - C_{jk})^2} \tag{4.1}$$

As distâncias dos exemplos de treinamento a cada centróide são utilizadas como entrada para as funções de base radial (RBF), as quais calculam a ativa-

[1] Por centróide deve-se entender o vetor que representa o centro de um determinado grupo.

ção de cada centróide à entrada. Este trabalho considera a Equação 4.2 para computar a ativação, a qual segue uma função Gaussiana. A variação no valor de σ afeta a dispersão da função. Quanto maior o valor de σ, mais padrões distantes são aceitos por um centróide. A Figura 4.1 apresenta a saída da função de ativação para $\sigma = 1$, $\sigma = 2$ e $\sigma = 3$.

$$Act(x, \sigma) = e^{-\frac{x^2}{2\sigma^2}} \tag{4.2}$$

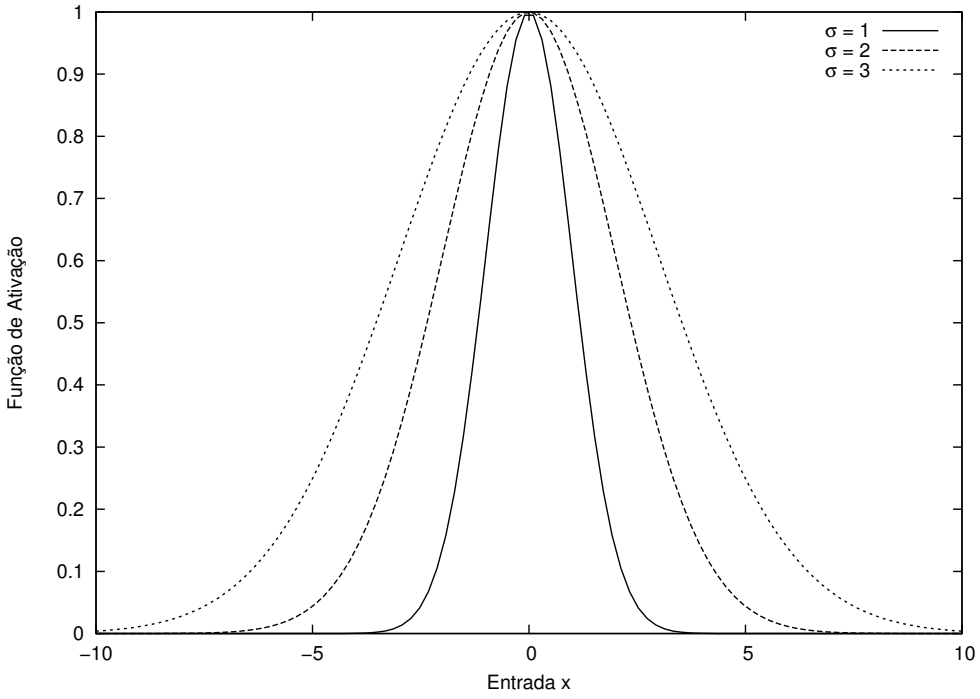


Figura 4.1: Função de Base Radial (RBF) para $\sigma = 1$, $\sigma = 2$ e $\sigma = 3$.

Após computar a ativação, o valor obtido é comparado com um nível de aceitação, denominado *threshold*. Esse parâmetro define o nível mínimo de ativação que um centróide precisa atingir para aceitar o padrão como pertencente a si. O *threshold* funciona como uma forma de saber se o padrão recebido está próximo o suficiente de um grupo, ou se é necessário criar um novo grupo que o contenha.

A ativação de todos os centróides é então calculada. Aquele que fornece a maior ativação é selecionado como o receptor do padrão de entrada. Isso significa que o padrão será agrupado por esse centróide. Caso nenhum centróide apresente ativação suficiente, cria-se um novo, tendo como centróide os valores do padrão de entrada.

Cada centróide é considerado como um estado da cadeia de Markov. As transições entre estados são armazenadas em uma matriz de probabilidades, que governa a chance de transição de cada estado para os demais.

O comportamento de um processo é modelado por meio de uma cadeia de Markov com os estados visitados durante sua execução. Exemplos de estados

são: escrevendo dados em disco, lendo dados do disco, alocando memória, etc.

As relações entre os estados visitados pelo processo são modeladas por meio das probabilidades associadas a transições. Por exemplo, considere um conjunto de dados que contenha os seguintes atributos: número de *bytes* lidos/escritos em disco (x_1), memória disponível (x_2) e espaço em disco disponível (x_3). Um estado s desse processo pode ser representado por um centróide $s = (x_1, x_2, x_3)$. Suponha que tenham sido encontrados dois estados mais representativos que o processo visita durante sua execução: $s_1 = (10, 100, 200)$ e $s_2 = (50, 80, 150)$. Considere, ainda, que a probabilidade de transição de s_1 para s_2 dá-se por $p(s_1, s_2) = 80\%$ e $p(s_2, s_1) = 20\%$. Isso está ilustrado na Figura 4.2.

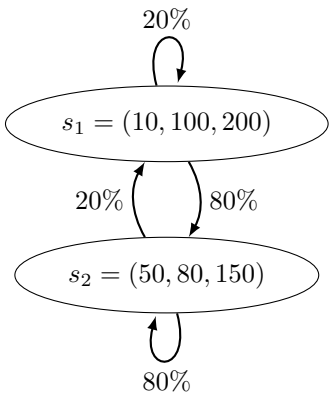


Figura 4.2: Exemplo de um processo modelado como uma cadeia de Markov.

No caso ilustrado na Figura 4.2, tem-se uma representação do comportamento normal do processo, modelado a partir de dados de treinamento. Desvios desse modelo normal são, consequentemente, apontados como faltas, ou transições inesperadas entre estados. Por exemplo, se o processo está atualmente no estado s_1, há 80% de chance de que ele vá para o estado s_2 e 20% de chance que permaneça em s_1. Caso seja feita uma transição para um outro estado, não esperado, considera-se que existe alto nível de novidade no comportamento desse processo. Outro indicativo de novidade se dá por uma mudança acentuada nas probabilidades de transições entre estados. A seguir, detalha-se como esse nível de novidade é medido.

Após obter a cadeia de Markov para os padrões de entrada apresentados, a Entropia do sistema é calculada de acordo com a Equação 4.3, a qual considera todas as probabilidades de transição. Nessa equação, $p(i, j)$ representa a probabilidade de transição do estado s_i para s_j, sendo c o número de centros. Assume-se $0 \cdot log_2(0) = 0$.

$$E_t = -\sum_{j=1}^{c}\sum_{i=1}^{c} p(i,j) \cdot log_2(p(i,j)) \tag{4.3}$$

Essa abordagem tem como saída o quadrado da variação da Entropia, en-

tre padrões de entrada consecutivos, como apresentado na Equação 4.4. Utilizando a exponenciação, valores negativos são evitados, a fim de obter um valor absoluto para a variação.

$$D_t = (E_t - E_{t-1})^2 \tag{4.4}$$

A visita a um novo estado significa que um centróide acabou de ser criado e certo nível de novidade é detectado. Isso causa um aumento na Entropia do sistema, que é refletido na saída da abordagem. O procedimento completo é apresentado no Algoritmo 1.

Algoritmo 1: Nível de Novidade

```
Sigma e Threshold são definidos pelo usuário.
T ← Threshold;
σ ← Sigma;
estadoAnterior ← ∅;
entropiaAnterior ← ∅;
for I in Entrada do
    ativacaoMaxima ← 0;
    for c in centroides do
        d ← distanciaEuclidiana(I, c);
        ativacao ← Act(d, σ);
        if ativacao > T and ativacao > ativacaoMaxima then
            ativacaoMaxima ← ativacao;
            centroideId ← c;
        end
    end
    if ativacaoMaxima == 0 then
        criaNovoCentroide(I);
        centroideId ← I;
    end
    atualizeMarkov(estadoAnterior, centroideId);
    entropiaAtual ← entropiaMarkov();
    Novidade ← (entropiaAtual − entropiaAnterior)^2;
    entropiaAnterior ← entropiaAtual;
    estadoAnterior ← centroideId;
    saida(Novidade);
end
```

4.3 Observações sobre a Atualização da Cadeia de Markov

A cadeia de Markov, utilizada para representar as transições entre estados de um processo, pode ser construída armazenando-se o número de transições que um processo fez para outros estados, a fim de calcular a probabilidade de

transição entre eles. Por exemplo, considere a cadeia apresentada na Tabela 4.1 que é representada pela Matriz 4.5.

Tabela 4.1: Exemplo de cadeia de Markov, que armazena o número de transições entre estados do processo.

De/Para	Estado s_1	Estado s_2
Estado s_1	50	1000
Estado s_2	1000	50

$$M = \begin{bmatrix} 50 & 1000 \\ 1000 & 50 \end{bmatrix} \tag{4.5}$$

Para o sistema apresentado, pode-se calcular a Entropia como mostrado na Equação 4.6.

$$E = -(2 \cdot (50/1050) \cdot \log_2(50/1050) + 2 \cdot (1000/1050) \cdot \log_2(1000/1050)) \simeq 0,55239 \tag{4.6}$$

O problema com essa abordagem é que, devido à existência de um número considerável de visitas para alguns estados, quando se adiciona um novo estado não há variação significativa na Entropia do sistema, mascarando a novidade ocorrida.

Considere como a adição de um estado afeta o valor da Entropia nesse sistema. A Matriz 4.3 apresenta um novo estado adicionado. O valor da Entropia para essa nova cadeia é $0,56305$, o que representa uma variação de aproximadamente $0,01$ no valor da Entropia do sistema.

$$M' = \begin{bmatrix} 50 & 1000 & 1 \\ 1000 & 50 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 4.3: Matriz referente à Figura 4.5 com um novo estado adicionado.

Com o intuito de intensificar o valor da variação de Entropia quando um novo estado é visitado, realizou-se a seguinte modificação no armazenamento da cadeia de Markov, proposta por Albertini (2009): ao invés de se armazenar o número de transições feitas entre estados, armazena-se somente valores de probabilidade, que são atualizados de acordo com uma média móvel exponencial ponderada. O fator de ponderação é denominado ψ. A Equação 4.7 apresenta o mecanismo de atualização da cadeia, sendo m o estado anterior e k o estado atual. Após essa atualização, normaliza-se o valor das probabilidades para que sua integral seja um.

$$M'_{ij} = \begin{cases} M_{i,j} + \psi, i = m, j = k \\ (1 - \psi) \cdot M_{i,j}, i = m, j \neq k \end{cases} \quad (4.7)$$

Considere o exemplo dado anteriormente da matriz apresentada na Figura 4.3. Utilizando essa modificação na atualização da cadeia, com $\psi = 0,5$, tem-se uma variação de Entropia igual a $0,9$ ao visitar o terceiro estado, intensificando o valor do nível de novidade detectado pela abordagem. A matriz obtida com essa modificação é apresentada na Figura 4.4.

$$M' = \begin{bmatrix} 0.00 & 0.33 & 0.67 \\ 1.0 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$

Figura 4.4: Matriz referente à Figura 4.5 com um novo estado adicionado e utilizando a modificação proposta.

Dessa forma, há uma atenuação no impacto causado por estados visitados muitas vezes ao longo do tempo. Com essa modificação, detecta-se um maior nível de variação quando se visita um novo estado, como desejado.

4.4 Análise de Complexidade da Abordagem

Foi feita uma análise da complexidade de tempo da abordagem no pior caso. Para isso, utilizou-se a notação *Big O* (Cormen et al., 2001). Essa notação pode ser empregada para analisar o crescimento assintótico de funções, a fim de encontrar um limite superior para a complexidade de execução de um algoritmo no pior caso.

O Algoritmo 2 apresenta, novamente, o cálculo do nível de novidade, juntamente com a análise de complexidade de cada linha do algoritmo. Conforme observado na listagem, o laço que calcula a distância de um padrão de entrada aos centróides, juntamente com a ativação deles, possui complexidade proporcional a $O(ck)$, onde c é o número de centros e k é o número de atributos de cada padrão (constante), o que resulta em uma complexidade linear.

Caso seja necessário criar um novo centro, deve-se reconstruir a matriz de transições com uma nova linha e uma nova coluna, o que possui custo $O(c^2)$, ou quadrático. De maneira semelhante acontece quando é feito o cálculo da Entropia do sistema, pois requer que a matriz inteira seja percorrida.

Portanto, levando em conta que a soma das complexidades dos laços internos é quadrática e que esse custo será imposto a cada um dos padrões de entrada apresentados, tem-se que a abordagem apresenta uma complexidade $O(n \cdot c^2)$ no pior caso, em que n é o número de exemplos de entrada e c é o número de centros criados.

É importante ressaltar, entretanto, que a variável de maior influência na complexidade do algoritmo é o número de centros, pois define o tamanho da matriz de probabilidades. Para que a abordagem possua boa capacidade de generalização, não deve haver um número explosivo de centros, logo espera-se que a busca por uma quantidade razoavelmente baixa de centros possa reduzir o custo da abordagem na prática, diminuindo o impacto do termo quadrático. Isso pode ser conseguido, principalmente, pela escolha adequada de valores para σ e *threshold*.

Algoritmo 2: Análise de Complexidade do Algoritmo para Nível de Novidade

```
Sigma e Threshold são definidos pelo usuário.
T ← Threshold // O(1);
σ ← Sigma // O(1);
estadoAnterior ← ∅ // O(1);
entropiaAnterior ← ∅ // O(1);
for I in Entrada do
    ativacaoMaxima ← 0 // O(1);
    for c in centroides do
        d ← distanciaEuclidiana(I, c) // O(k);
        ativacao ← Act(d, σ) // O(1);
        if ativacao > T and ativacao > ativacaoMaxima then
            centroideId ← c // O(1);
            ativacaoMaxima ← ativacao // O(1);
        end
    end
    if ativacaoMaxima == 0 then
        criaNovoCentroide(I) // O(c²) quadrático, envolve atualizar toda
        matriz de transicoes;
        centroideId ← I // O(1);
    end
    atualizeMarkov(estadoAnterior, centroideId) // O(c) linear, envolve
    percorrer uma linha da matriz de transicoes;
    entropiaAtual ← entropiaMarkov() // O(c²) quadrático, envolve
    percorrer toda matriz de transicoes;
    Novidade ← (entropiaAtual − entropiaAnterior)² // O(1);
    entropiaAnterior ← entropiaAtual // O(1);
    estadoAnterior ← centroideId // O(1);
    saida(Novidade) // O(1);
end
```

4.5 Considerações finais

Neste capítulo foi apresentada a abordagem proposta para detecção de faltas, a qual leva em conta o comportamento de processos. Mostrou-se como a técnica associa conceitos de funções de base radial, Teoria da Informação e

Cadeias de Markov para detectar faltas. No capítulo seguinte são apresentados experimentos realizados utilizando a abordagem proposta sobre bases de dados da literatura e geradas com injenção de faltas.

CAPÍTULO

5

Experimentos

5.1 Considerações iniciais

Neste capítulo são apresentados experimentos realizados com a abordagem proposta. Os resultados obtidos até o momento indicam que a abordagem pode realizar detecção de faltas de maneira mais eficaz do que as demais abordagens estudadas.

5.2 Medidas utilizadas para avaliação

Nesta seção, as medidas utilizadas para avaliação da técnica proposta são apresentadas. Sete medidas foram escolhidas para esse propósito (Witten e Frank, 2005). Nas equações, VP representa verdadeiros positivos, VN verdadeiros negativos, FP falsos positivos e FN falsos negativos.

Precisão (*Precision*) Taxa que mede a razão de exemplos verdadeiramente positivos dos que foram classificados como positivos:

$$\text{Precisão} = \frac{VP}{VP + FP} \tag{5.1}$$

Revocação (*Recall*) Taxa que mede a razão de exemplos classificados como positivos dos que realmente são positivos:

$$\text{Revocação} = \frac{VP}{VP + FN} \tag{5.2}$$

Especificidade (*Specificity*) Taxa que mede a razão de exemplos classificados como negativos dos que realmente são negativos:

$$\text{Especificidade} = \frac{VN}{VN + FP} \tag{5.3}$$

F-Measure É uma média harmônica entre a precisão e revocação:

$$\text{F-Measure} = \frac{2}{1/\text{Precisão} + 1/\text{Revocação}} \tag{5.4}$$

Acurácia Mede a taxa de exemplos corretamente classificados do número total de exemplos:

$$\text{Acurácia} = \frac{VP + VN}{VP + FP + FN + VN} \tag{5.5}$$

Taxa de Falsos Positivos (Erro Tipo I) Mede a taxa de exemplos classificados erroneamente como positivos do total de exemplos realmente negativos:

$$\text{TFP} = \frac{FP}{FP + VN} \tag{5.6}$$

Taxa de Falsos Negativos (Erro Tipo II) Mede a taxa de exemplos classificados erroneamente como negativos do total de exemplos realmente positivos:

$$\text{TFN} = \frac{FN}{VP + FN} \tag{5.7}$$

5.3 Experimentos iniciais

Nesta seção são apresentados os primeiros experimentos realizados utilizando a abordagem proposta. Inicialmente, um conjunto de dados sintético foi gerado para avaliar a detecção de anomalias, que serão associadas posteriormente a faltas. A seguir, dados utilizados comumente na literatura, provenientes do repositório UCI[1], foram empregados para avaliar a abordagem sobre conjuntos de dados bem conhecidos. A Tabela 5.1 apresenta um sumário sobre as bases (conjuntos) de dados utilizadas.

Antes de cada experimento ser realizado, os dados foram normalizados para que ficassem no intervalo $[0, 1]$. Todas as instâncias com campos faltantes foram removidas dos conjuntos de dados. Os experimentos utilizando

[1]Os dados podem ser obtidos em: http://archive.ics.uci.edu/ml/. Último acesso: 01/10/2009.

dados do UCI foram executados 120 vezes, para que fosse garantida significância estatística, de acordo com o Teorema do Limite Central (Walpole et al., 2006). Para esses dados, a classe anômala foi considerada positiva e a normal negativa, tornando-se um problema de classificação binária.

Foram calculadas medidas de eficácia do classificador, tais como Acurácia, Precisão, Revocação, Especificidade, Taxa de Falsos Positivos, Taxa de Falsos Negativos e *F-measure*. Para cada uma dessas medidas foi computado o máximo, mínimo, primeiro e terceiro quartis, mediana, média e desvio padrão.

A abordagem foi treinada com $2/3$ das instâncias da classe negativa. As instâncias restantes ($1/3$) foram utilizadas para teste, juntamente com os exemplos da classe positiva. Após a fase de treinamento nenhum novo centróide foi criado, nem a cadeia de Markov atualizada. Se uma instância foi classificada como pertencente a um dos centróides criados durante o treinamento, então era considerada da classe negativa (normal), caso contrário da classe positiva (anômala).

A técnica de validação cruzada (*cross-validation*) não foi empregada na avaliação, pois ela modifica a ordem dos exemplos observados. Isso afeta o comportamento da abordagem proposta, já que ela visa modelar relações temporais existentes entre observações. A quebra em dois conjuntos de $2/3$ e $1/3$ dos dados foi feita de forma a manter o maior número de exemplos contíguos no mesmo conjunto.

Tabela 5.1: Bases de dados utilizadas nos experimentos iniciais.

Base	**Fonte**	**Classe normal**	**Classe Anômala**
Senóide	Gerada sinteticamente	$-1 \leq x \leq 1$, 99%	$x > 1$, 1%
Iris	Repositório UCI	*setosa*, 33%	*virginica* e *versicolor*, 67%
Breast Cancer	Repositório UCI	Câncer benigno, 63%	Câncer maligno, 37%
Biomed	Repositório UCI	Pacientes saudáveis, 40%	Pacientes doentes, 60%

5.3.1 Experimento com dados sintéticos de uma senóide

Neste experimento foi avaliada a capacidade básica de detecção de anomalias da abordagem proposta. Para isso, foi utilizada uma senóide, a fim de criar circunstâncias de modelagem simples, pelo fato da senóide ser uma função periódica e com comportamento bem definido. Nesse contexto, foram gerados 1000 pontos da senóide, sendo que em dez deles foram introduzidas anomalias, no caso valores maiores do que 1, fora do conjunto imagem da função seno ($[-1, 1]$). A abordagem foi executada com parâmetros $\sigma = 1.0$ e *threshold* = 0.05. Para esse experimento, ao contrário dos outros com as bases UCI, foi permitido que a cadeia de Markov fosse atualizada constantemente, bem como novos centróides fossem criados, já que a abordagem foi executada de maneira *online*, sem fase de treino e teste. A Figura 5.1 apresenta a saída

da abordagem e indica que todas as dez anomalias foram detectadas. A Figura 5.2 ilustra os centróides a que cada padrão foi atribuído.

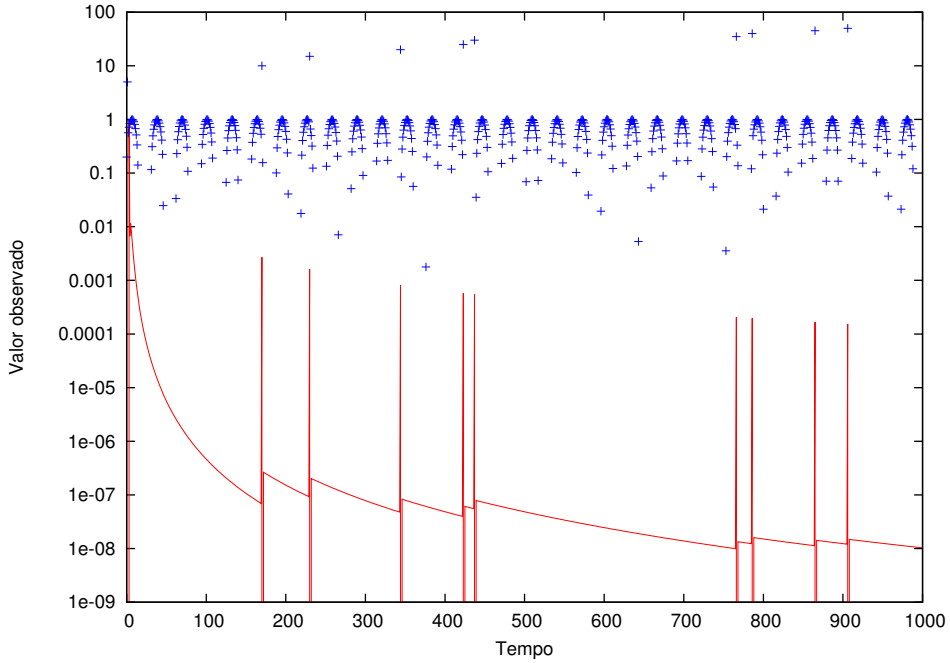


Figura 5.1: A curva representa o nível de novidade do algoritmo (escala logarítmica no eixo y) e os pontos a senóide simulada. Os valores de seno acima de 1 são as dez anomalias inseridas.

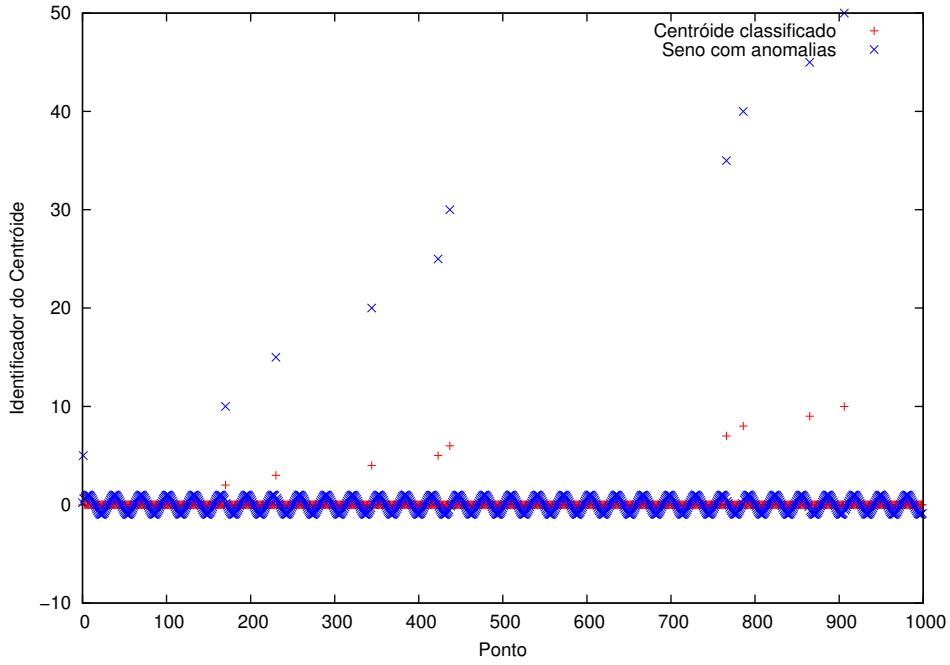


Figura 5.2: Identificador do centróide a que cada padrão foi atribuído.

5.3.2 *Experimento com a base Iris*

O segundo experimento foi realizado para avaliar a abordagem proposta sobre dados bem conhecidos, notadamente os da base Iris, criada por Fisher

(1936). Essa base classifica flores Iris em três espécies: *setosa, virginica* e *versicolor*. Quatro atributos foram medidos para realizar a discriminação: comprimento e largura das sépalas e das pétalas. Nesse experimento, a classe negativa foi considerada como a espécie *setosa*. Flores das espécies *virginica* e *versicolor* foram consideradas anomalias (classe positiva). Os resultados obtidos são apresentados na Tabela 5.2. Como pode ser observado, foi possível atingir um alto valor de *F-measure*.

Tabela 5.2: Resultados para base Iris – Threshold = $0,5$ e $\sigma = 0,5$

Medida	Min.	1o Qu.	Mediana	Média	3o Qu.	Máx.	Desvio P.
Acurácia	0,9655	0,9914	0,9914	0,9912	1,0000	1,0000	0,0101
Precisão	0,9615	0,9901	0,9901	0,9900	1,0000	1,0000	0,0113
Revocação	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	0,0000
Especificidade	0,7500	0,9375	0,9375	0,9359	1,0000	1,0000	0,0735
Taxa de Falsos Positivos (Erro Tipo I)	0,0000	0,0000	0,0625	0,0640	0,0625	0,2500	0,0735
Taxa de Falsos Negativos (Erro Tipo II)	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
F-Measure	0,9804	0,9950	0,9950	0,9949	1,0000	1,0000	0,0057

Tabela 5.3: Resultados para base *Breast Cancer* – Threshold = $0,5$ e $\sigma = 0,5$

Medida	Min.	1o Qu.	Mediana	Média	3o Qu.	Máx.	Desvio P.
Acurácia	0,8682	0,8760	0,8811	0,8880	0,9018	0,9173	0,0156
Precisão	0,9520	0,9589	0,9704	0,9678	0,9757	0,9849	0,0101
Revocação	0,8075	0,8243	0,8326	0,8471	0,8745	0,9121	0,0281
Especificidade	0,9257	0,9392	0,9595	0,9541	0,9662	0,9797	0,0154
Taxa de Falsos Positivos (Erro Tipo I)	0,0202	0,0337	0,0405	0,0458	0,0608	0,0743	0,0154
Taxa de Falsos Negativos (Erro Tipo II)	0,0878	0,1255	0,1674	0,1529	0,1757	0,1925	0,0281
F-Measure	0,8833	0,8914	0,8978	0,9031	0,9170	0,9316	0,0148

Tabela 5.4: Resultados para base Biomed – Threshold = $0,1$ e $\sigma = 0,1$

Medida	Min.	1o Qu.	Mediana	Média	3o Qu.	Máx.	Desvio P.
Acurácia	0,5872	0,6147	0,6330	0,6459	0,6789	0,7248	0,0405
Precisão	0,6620	0,7033	0,7143	0,7227	0,7606	0,7895	0,0362
Revocação	0,6119	0,6418	0,6716	0,6898	0,7313	0,8060	0,0497
Especificidade	0,4286	0,5417	0,5714	0,5758	0,6250	0,7143	0,0718
Taxa de Falsos Positivos (Erro Tipo I)	0,2857	0,3750	0,4286	0,4242	0,4583	0,5714	0,0718
Taxa de Falsos Negativos (Erro Tipo II)	0,1940	0,2687	0,3284	0,3102	0,3582	0,3881	0,0497
F-Measure	0,6565	0,6769	0,6923	0,7050	0,7258	0,7826	0,0356

5.3.3 Experimento com a base Breast Cancer

Este experimento considerou a base *Breast Cancer*, disponível no repositório UCI. Essa base consiste de pacientes com câncer maligno ou benigno. Os atributos consistem de informações como idade, tamanho do tumor, e outras[2].

[2]A descrição completa da base encontra-se no repositório UCI: http://archive.ics.uci.edu/ml/datasets/Breast+Cancer. Último acesso: 9 de novembro de 2009.

Para esse experimento, considerou-se que a classe positiva (anômala) era a de pacientes com câncer maligno e a negativa como pacientes com câncer benigno. Os resultados obtidos são apresentados na Tabela 5.3. Eles também confirmam a eficácia da abordagem proposta, com um *F-measure* médio de aproximadamente 90% e erros tipo I e II abaixo de 9%.

5.3.4 Experimento com a base Biomed

Este experimento foi realizado com a base Biomed, disponível no repositório UCI. Ela é dividida em duas classes: pessoas com medições de sangue normais, as quais foram assumidas como da classe negativa, e outras que são portadoras de uma doença rara, consideradas da classe anômala (positiva).

Foram utilizados somente os seguintes atributos da base nesse experimento: idade dos pacientes e as quatro medições de sangue disponíveis. Os outros atributos eram identificadores e não se relacionavam com a tarefa de aprendizado.

Nessas circunstâncias, os resultados obtidos não foram tão bons quanto os obtidos com as outras bases (Tabela 5.4). Acredita-se que isso se deve à distribuição dos dados, que não são claramente divididos entre normais e portadores, além da quantidade pequena de exemplos. Entretanto, a abordagem atingiu um *F-measure* médio de 70%, com desvio padrão de $0,03$, o que é melhor do que a rede neural SONDE pôde atingir, com um *F-measure* médio de $0,59$ e desvio padrão de $0,04$ (Albertini e de Mello, 2008).

5.4 Experimentos de detecção de faltas com utilitários UNIX

Foram realizados experimentos para avaliar a eficácia da abordagem proposta na detecção de faltas de processos reais. Para isso, foram selecionados utilitários UNIX e faltas foram injetadas neles via *software*. Nesta seção esses experimentos são descritos.

Existem três abordagens básicas para o estudo de sistemas em ambientes com faltas: modelagem analítica, técnicas experimentais e injeção de erros e faltas (Kanawati et al., 1995). O problema com a modelagem analítica é que os modelos podem se tornar complexos demais e, caso sejam feitas simplificações, os resultados podem não ser satisfatórios. Técnicas experimentais envolvem coletar dados do sistema até que ele falhe. Essa abordagem pode ser problemática e cara, pois envolve fazer com que o sistema real venha a falhar.

Injeção de faltas e erros é uma outra abordagem para estudar o comporta-

mento de sistemas em ambientes com faltas. A injeção de faltas via *hardware*, por meio da injeção a nível de pinos ou com radiação de íons pesados, pode ser cara e causar danos físicos ao equipamento sendo estudado (Hsueh et al., 1997).

A técnica de injeção via *software*, também conhecida por *Software Implemented Fault Injection* (SWIFI) (Kanawati et al., 1995) não requer *hardware* específico e custoso para injetar faltas. Além disso, a técnica pode ser usada para afetar aplicações específicas, o que é difícil de ser conseguido com injeção via *hardware*. Portanto, como este trabalho visa analisar o comportamento de processos no espaço de usuário, foi escolhida a técnica de injeção via *software*.

Para injetar faltas foi utilizada a chamada de sistema `ptrace` do UNIX (Padala, 2002), a qual permite a interceptação de chamadas de sistema e modificação das mesmas no espaço de usuário. É possível iniciar um processo em modo *trace* e pará-lo a cada entrada e saída de todas as chamadas de sistema realizadas, inspecionando e alterando os valores contidos nos registradores no momento. Isso permite capturar o comportamento do processo durante sua execução, observando as chamadas que ele realiza e quais argumentos e retornos são passados. Essa abordagem também apresenta a vantagem de não necessitar do código fonte da aplicação. É possível realizar a interceptação possuindo somente o binário, não importando a linguagem na qual o programa foi escrito.

Foram selecionados três utilitários UNIX para análise: `ls`, `cat` e `grep`. Eles foram escolhidos pois realizam tarefas diferentes, mas comuns a vários usuários, como listar diretórios (`ls`), concatenar arquivos (`cat`) e buscar padrões (`grep`) contidos em arquivos. Esses utilitários estão entre as vinte ferramentas mais usadas em um sistema UNIX, de acordo com Hanson et al. (1984). Os utilitários foram executados sobre os arquivos do código fonte do Linux[3], versão 2.6.30.1. A Tabela 5.5 sumariza informações sobre a divisão em classes dos dados.

Dois experimentos foram realizados para analisar como os utilitários se comportavam em um ambiente com faltas. O primeiro considerou a abordagem proposta, utilizando funções RBF para agrupar o comportamento e gerar uma cadeia de Markov, representativa dos estados do processo. O segundo experimento considerou a dissimilaridade dos dados extraídos, utilizando a medida de distância *Dynamic Time Warping*.

Foram avaliados três cenários em cada experimento: 1) a execução em um ambiente sem faltas sobre um conjunto de arquivos; 2) a execução em um ambiente com faltas sobre o mesmo conjunto de arquivos e 3) a execução em um ambiente sem faltas sobre um conjunto diferente de arquivos.

[3]O código pode ser obtido em http://www.kernel.org. Último acesso: 10 de novembro de 2009.

Tabela 5.5: Bases geradas para os utilitários `cat`, `ls` e `grep`

Utilitário	Classe normal	Classe anômala (faltas)
`ls`	77%	23%
`cat`	81%	19%
`grep`	80%	20%

5.4.1 Análise com a abordagem proposta

Em um primeiro momento, cada processo foi iniciado de maneira instrumentada pela chamada `ptrace`. Na primeira rodada os processos foram executados em um ambiente sem faltas, a fim de capturar todas as chamadas de sistema realizadas. A análise do histograma das chamadas de cada processo forneceu a base para a seleção de pontos de inserção de faltas.

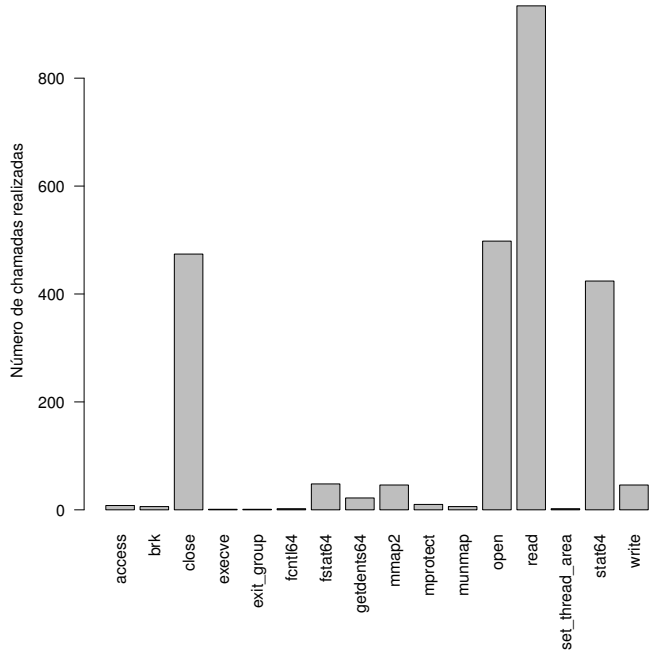
A Figura 5.3(a) apresenta o histograma das chamadas de sistema realizadas pelo processo `grep` em uma execução sem faltas. Os histogramas consideram o número de entradas e saídas de cada chamada de sistema realizada. A chamada `read` foi a mais utilizada, portanto foi selecionada como ponto de inserção durante a execução com faltas. Isso foi feito de forma que a cada chamada `read` que tentava transferir mais de 4096 *bytes* não conseguisse transferir nada. Esse efeito foi produzido por meio da alteração de valores de registradores, utilizando a *flag* `PTRACE_SETREGS` da chamada `ptrace`.

Foi possível observar uma mudança proeminente no número de chamadas `read` para `write`. A diminuição no número de chamadas `read` ocorre devido à inserção de faltas, o que levou também chamadas subsequentes a não serem concluídas, quando as primeiras não conseguiam transferir dados. O aumento das chamadas `write` é explicado devido às mensagens de erro geradas pelo processo, as quais foram escritas na saída padrão (*stdout*).

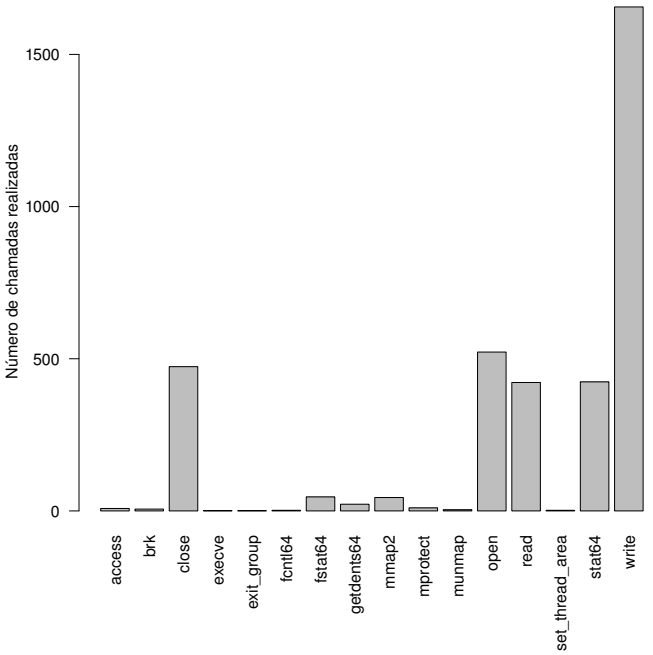
A seguir, o processo `grep` foi instrumentado para capturar somente chamadas `read/write`, juntamente com o número de *bytes* transferidos em cada chamada. Os dados foram executados com a abordagem proposta, que agrupou comportamento similar em estados de uma cadeia de Markov, a qual é exibida na Figura 5.4(a) para a execução sem faltas.

Analisando ambas as cadeias de Markov apresentadas na Figura 5.4, é possível observar uma mudança de comportamento substancial no ambiente com faltas. Enquanto em uma execução normal havia 10% de chance do processo permanecer no estado 'escrevendo poucos dados', em uma execução com faltas houve 97% de probabilidade disso ocorrer. Observando essas características, é possível distinguir entre uma execução com faltas e outra sem.

A abordagem foi utilizada para calcular níveis de Entropia dos processos.

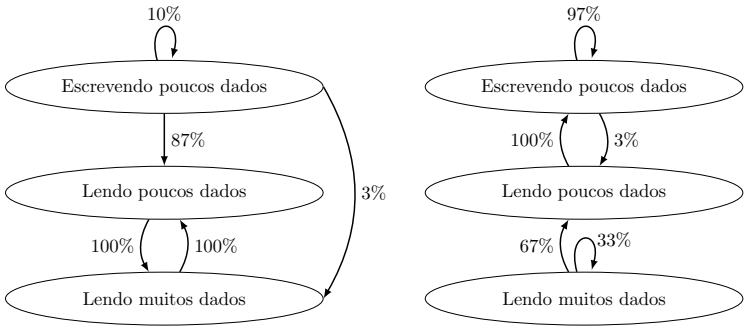


(a) Ambiente sem faltas.



(b) Ambiente com faltas.

Figura 5.3: Histogramas para execuções do `grep`.



(a) Ambiente sem faltas. (b) Ambiente com faltas.

Figura 5.4: Cadeias de Markov para execuções do `grep` (considerando somente chamadas `read/write`.

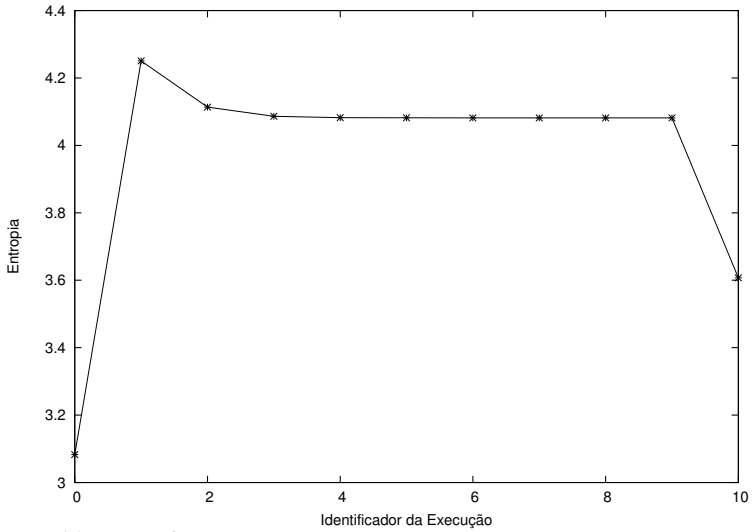
Uma série foi construída concatenando dez execuções normais de cada processo à execução com faltas ao final. O nível de Entropia foi medido após cada execução ser apresentada a abordagem. A Figura 5.5 mostra esses resultados.

É possível observar que o sistema estabiliza seu nível de Entropia após algumas execuções corretas. As execuções de 0 a 9 correspondem a ambientes sem faltas. A execução 10 é a realizada com faltas injetadas. Todos os gráficos apresentam uma mudança de Entropia após a execução em um ambiente com faltas.

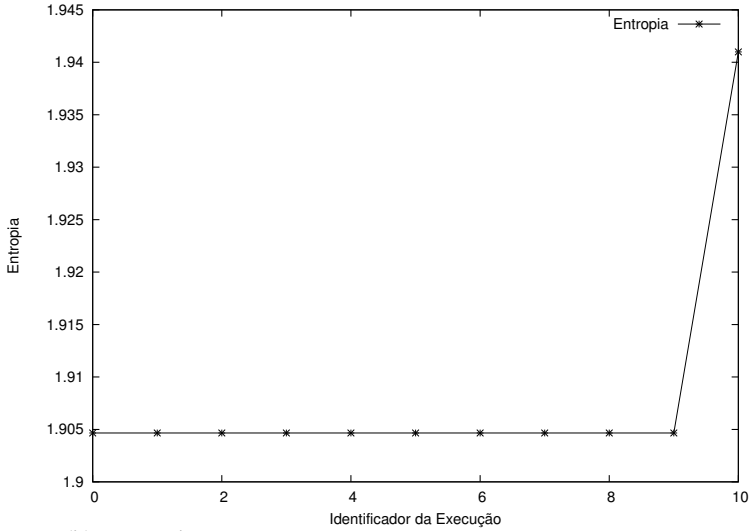
O próximo experimento utilizou a medida de distância *Dynamic Time Warping* e confirmou que o comportamento dos processos manteve-se muito similar, mesmo para execuções sobre arquivos diferentes, em ambientes sem faltas. Já quando executado em um cenário com faltas, a distância medida entre as séries aumentou significativamente, o que corrobora os resultados obtidos com a abordagem proposta. Esses resultados são apresentados a seguir.

5.4.2 Análise de resultados obtidos utilizando Dynamic Time Warping

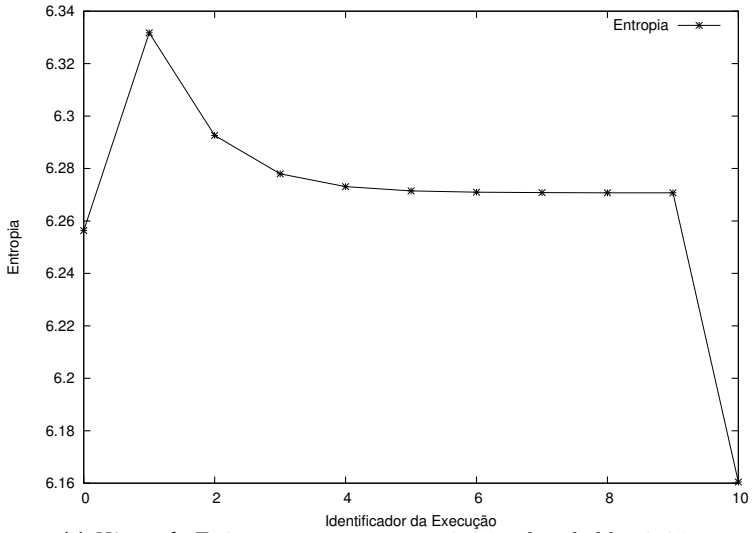
Além da análise utilizando a abordagem proposta para encontrar níveis de novidade e estados representativos de processos, buscou-se comparar as séries extraídas de ambientes com e sem faltas. Esperou-se que séries provenientes de ambientes sem faltas fossem mais similares entre si do que se comparadas a séries de ambientes com faltas. Para computar a dissimilaridade entre duas séries pode-se utilizar medidas de distância. Com esse intuito, foi realizado um experimento utilizando a medida de distância *Dynamic Time Warping* (DTW) (Keogh e Ratanamahatana, 2005). Essa medida é mais robusta para cálculo de distâncias entre séries temporais do que a distância Euclidiana, pois permite que as séries sejam analisadas mesmo se não estive-



(a) Níveis de Entropia para o `grep`, $\sigma = 0,1$ e $threshold = 0,1$.



(b) Níveis de Entropia para o `cat`, $\sigma = 0,8$ e $threshold = 0,8$.



(c) Níveis de Entropia para o `ls`, $\sigma = 0,05$ e $threshold = 0,05$.

Figura 5.5: Níveis de Entropia para dez execuções normais e uma execução com faltas.

Tabela 5.6: Distâncias DTW para execuções do `grep`.

	Normal 1	**Normal 2**	**Falta**
Normal 1	0	0,01	0,08
Normal 2	0,01	0	0,09
Falta	0,08	0,09	0

rem sincronizadas no eixo do tempo. Essa medida foi utilizada para computar a distância entre séries extraídas das execuções em ambientes com e sem faltas.

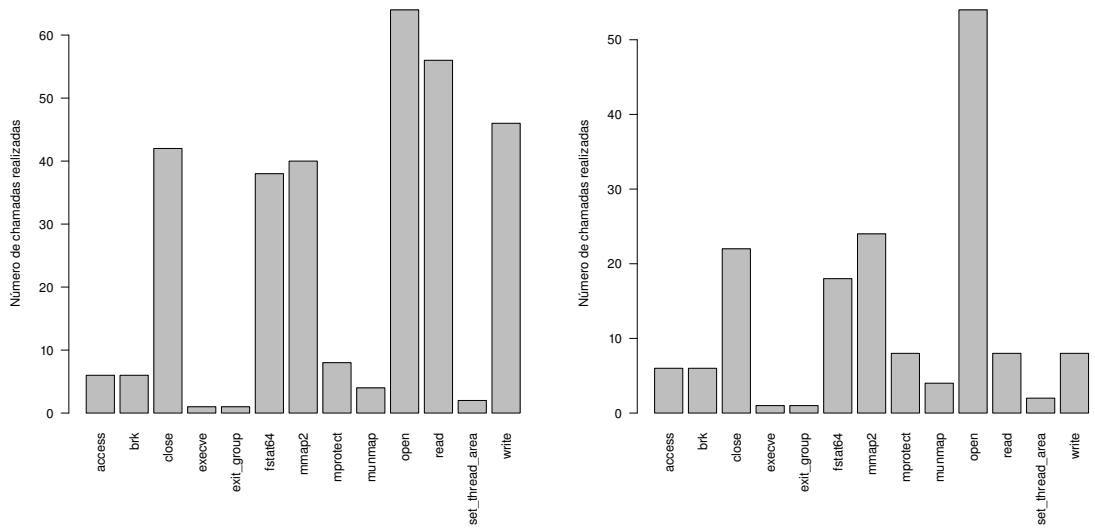
A técnica DTW (Keogh e Ratanamahatana, 2005) visa alinhar duas séries no eixo do tempo, encontrando o melhor ponto de sincronização entre elas, a fim de computar a menor distância. Considere duas séries $X = x_0, x_1, \ldots, x_{m-1}$ e $Y = y_0, y_1, \ldots, y_{n-1}$, de tamanhos m e n respectivamente. Inicialmente cria-se uma matriz $D_{m,n}$ em que cada elemento (i, j) representa a distância Euclidiana entre o elemento i de X e j de Y, ou seja, $D_{i,j} = (X_i - Y_j)^2$. Posteriormente, é criada uma nova matriz D' que acumula a distância total entre cada par de pontos das duas séries. Após esse cálculo, a distância entre as séries é computada somando o caminho mais curto começando na parte inferior direita da matriz até a superior esquerda. Esse caminho representa a melhor sincronização entre as duas séries. A soma de seus elementos é a distância DTW.

Inicialmente, a técnica DTW foi executada para o processo `grep`. As Figuras 5.3(a) e 5.3(b) apresentam os histogramas das execuções em ambientes diferentes. A Tabela 5.6 apresenta os resultados para as distâncias calculadas utilizando DTW[4] (Tormene et al., 2009). Foi possível observar que a distância é menor entre séries geradas a partir de execuções em ambientes sem faltas, do que se comparadas com séries de ambientes com faltas.

A seguir, foi estudado o comportamento do utilitário `cat`. O histograma da Figura 5.6(a) apresenta as chamadas realizadas durante a execução sem faltas. Observou-se que a chamada `open` foi uma das mais empregadas pelo processo, portanto decidiu-se utilizá-la como ponto de inserção de faltas. Após permitir que quinze chamadas `open` fossem realizadas com sucesso, os valores dos registradores foram alterados, durante as chamadas, para conter endereços espúrios dos arquivos que deveriam ser abertos. Como consequência, todas as chamadas `open` subsequentes falharam.

Foi então obtido o histograma apresentado na Figura 5.6(b), no qual pode ser observada a forte queda no número de chamadas `read` e `write`. Ambas quedas devem-se ao fato de o processo `cat` não ter conseguido obter os descritores de arquivos pela chamada `open`, o que tornou impossível lê-los e concatená-los. Os resultados das distâncias computadas utilizando DTW

[4]O cálculo foi realizado utilizando o pacote DTW do *software* estatístico R.



(a) Ambiente sem faltas.

(b) Ambiente com faltas.

Figura 5.6: Histogramas para execuções do `cat`.

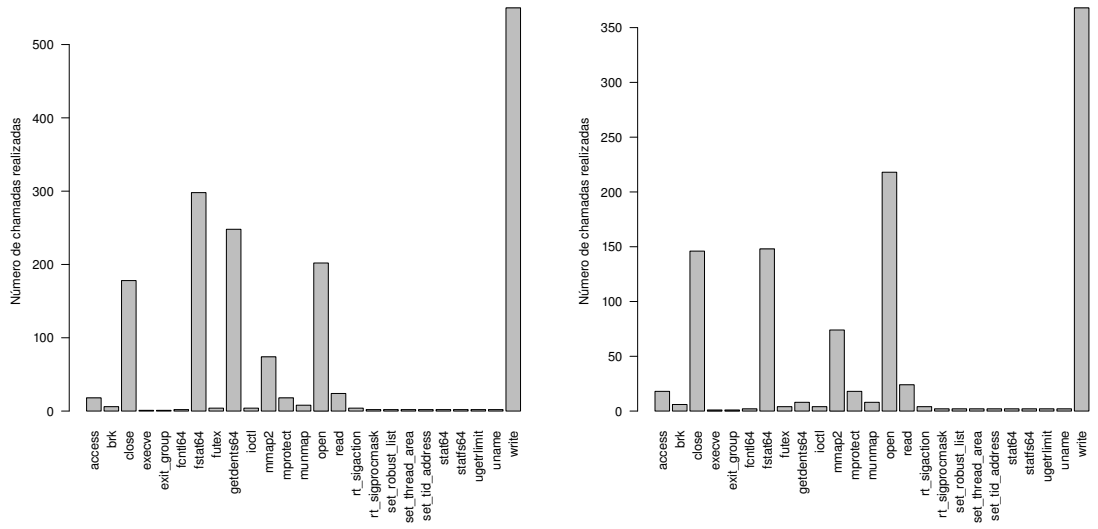
Tabela 5.7: Distâncias DTW para execuções do `cat`.

	Normal 1	**Normal 2**	**Falta**
Normal 1	0	0	6,26
Normal 2	0	0	6,26
Falta	6,26	6,26	0

são apresentados na Tabela 5.7. A distância zero entre diferentes execuções sem faltas pode ser explicada pelo fato das séries terem sido reduzidas a um mesmo tamanho, o que foi necessário para computar sua distância DTW, e pode ter causado ambas a ficarem com os mesmos elementos.

O último utilitário analisado nesse experimento foi o `ls`. O histograma da Figura 5.7(a) apresenta as chamadas de sistema realizadas durante uma execução sem faltas. Foi possível observar que a chamada `fstat64` foi uma das mais empregadas, além de chamadas `write` para escrever os resultados na saída padrão. A chamada `fstat64` é utilizada para obter informações sobre um arquivo, tais como ID do dispositivo que contém o arquivo, número de *inode*, ID do dono, e outras informações. Essa chamada foi escolhida para inserção das faltas. Após 60 chamadas com sucesso, os valores nos registradores foram alterados para conter um valor espúrio de endereço, em que as informações sobre o arquivo deveriam ser armazenadas.

O histograma da execução em um ambiente com faltas é apresentado na Figura 5.7(b), que mostra uma substancial queda no número de chamadas `getdents64`, usadas para obter informações sobre diretórios. Isso ocorre devido à impossibilidade de identificar se um arquivo era um diretório, informação retornada pela chamada `fstat64`. Os resultados computados com DTW são apresentados na Tabela 5.8, a qual confirma a similaridade entre execu-



(a) Ambiente sem faltas.

(b) Ambiente com faltas.

Figura 5.7: Histogramas para execuções do `ls`.

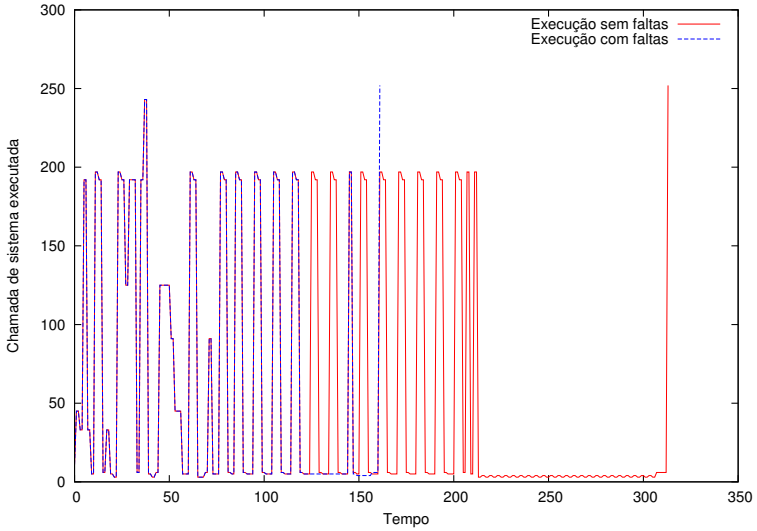


Figura 5.8: Exemplos de série do `cat`.

ções em ambientes sem faltas, bem como a maior distância entre séries de execuções com faltas e as sem faltas.

Foi possível observar pelos resultados obtidos que a inserção de faltas em diferentes utilitários os afetou em diferentes graus, conforme calculado pelas distâncias DTW. Além disso, os resultados também auxiliam a corroborar a variação de comportamento detectada pela abordagem proposta. A Figura 5.8 apresenta um exemplo da série do `cat` para uma execução sem faltas e outra com injeção de faltas.

A seção seguinte apresenta experimentos realizados com injeção de faltas em um servidor *Web*.

Tabela 5.8: Distâncias DTW para execuções do `ls`.

	Normal 1	**Normal 2**	**Falta**
Normal 1	0	0,79	3,65
Normal 2	0,79	0	3,35
Falta	3,65	3,35	0

5.5 Experimentos com o servidor Web Lighttpd

Neste experimento a abordagem proposta foi avaliada para detectar faltas injetadas no servidor *Web* Lighttpd[5], cujo código é disponibilizado abertamente. Esse servidor é utilizado em *sites* como *YouTube*[6], *Wikipedia*[7] e *Meebo*[8].

O gerador de carga *Apache Benchmark* (AB) foi utilizado para simular requisições de usuário durante a execução do servidor Lighttpd. Foram capturadas as seguintes informações sobre o processo do Lighttpd durante sua execução: valores de registradores das chamadas `read`, bem como dados fornecidos pela ferramenta vmstat, tais como número de chaveamentos de contexto por segundo, memória livre disponível, uso de *swap*, e outras[9]. As informações coletadas, além das específicas do processo, estão de acordo com o que é utilizado na literatura (Li et al., 2002).

Foram realizadas 30 execuções do experimento, a fim de obter significância estatística (Walpole et al., 2006). Séries temporais multivariadas foram geradas, as quais representavam o comportamento do processo ao longo do tempo. A última coluna do *data set* consistia de um valor binário, que representava se uma falta havia sido injetada (1) ou não (0).

Intervalos entre faltas, que determinaram o momento de injeção, foram gerados seguindo uma distribuição Poisson com taxa média de ocorrência $\lambda = 5$. As faltas foram injetadas modificando valores de registradores quando do retorno das chamadas `read`. Em média, cada *trace* gerado continha 20% de exemplos com falta e 80% de instâncias com comportamento normal.

Foi medido o F-measure atingido pela abordagem proposta, bem como das técnicas SVM (*Support Vector Machines*) e ARIMA (*Auto-Regressive Integrated Moving Average*). Essas técnicas foram escolhidas, a fim de comparação, por serem as que têm apresentado os melhores resultados, em geral, nas áreas de aprendizado de máquina e estatística (Hoffmann et al., 2007; Lorena e Carvalho, 2007; Xue et al., 2007). Os dados foram divididos da seguinte forma: $2/3$ foram utilizados para treinamento das técnicas e o $1/3$ restante para valida-

[5]O código fonte do servidor pode ser obtido em http://www.lighttpd.net/. Último acesso: 10 de novembro de 2009.

[6]www.youtube.com. Último acesso em: 10 de novembro de 2009.

[7]www.wikipedia.org. Último acesso em: 10 de novembro de 2009.

[8]www.meebo.com. Último acesso em: 10 de novembro de 2009

[9]Uma listagem completa das informações fornecidas pela ferramenta pode ser obtida em http://linux.die.net/man/8/vmstat. Último acesso: 10 de novembro de 2009.

ção. Foram utilizadas as implementações de SVM e ARIMA dos pacotes e1071 e stats, respectivamente, do *software* estatístico R. Essas implementações foram usadas também para estimação de parâmetros dessas técnicas.

Um teste de hipótese foi realizado para verificar se houve melhora estatisticamente significativa no F-measure atingido, ao comparar a abordagem com SVM e ARIMA. Para a comparação com SVM tem-se as hipóteses:

$$\begin{cases} H_0 : F_{\text{Abordagem}} \leq F_{\text{SVM}} \\ H_a : F_{\text{Abordagem}} > F_{\text{SVM}} \end{cases}$$

As hipóteses constituem um teste monocaudal direito. O teste foi realizado com nível de significância (erro tipo I) $\alpha = 0,01$, a fim de diminuir consideravelmente a chance de erro tipo I, ou seja, rejeitar a hipótese nula (H_0) quando ela é verdadeira. Neste experimento o erro tipo I consiste em afirmar que existem evidências suficientes para dizer que a abordagem atinge *F-Measure* maior do que SVM, quando na realidade não atinge.

Como o tamanho amostral n do experimento é igual a 30, foi empregada a estatística teste padronizada z e σ foi aproximado como o desvio padrão amostral s:

$$z = \frac{\overline{x} - \mu}{\sigma/\sqrt{n}} \tag{5.8}$$

Os valores obtidos nos experimentos estão listados na Tabela 5.9.

Tabela 5.9: Resultados Experimento *Lighttpd*.

Medida	**Valor**
Média F-Measure SVM	0,1925919
Média F-Measure Abordagem	0,6616754
Desvio Padrão Abordagem	0,2759962
Desvio Padrão SVM	0,1845472
Tamanho Amostral (n)	30

Substituindo os valores observados na Equação 5.8:

$$z = \frac{0,6616754 - 0,1925919}{0,2759962/\sqrt{30}} \simeq 9,309102$$

Tem-se que a área sob a curva à direita do valor de z encontrado é aproximadamente zero. Logo, é possível rejeitar a hipótese nula H_0 com um *p-value* $\simeq 0$ e $\alpha = 1\%$, e concluir que há evidências suficientes de que o F-measure obtido pela abordagem é maior do que o alcançado com o uso de SVM.

Aplicou-se então o modelo ARIMA no *data set* gerado da seguinte forma: foi realizada uma busca pela melhor ordem, dentro de um espaço restrito, por um modelo ARIMA que se ajustasse a cada coluna do *data set*. Da mesma

forma que para as outras técnicas, dois terços dos dados foram utilizados para ajustar o modelo e um terço para validação.

A detecção de faltas foi realizada considerando um intervalo de confiança de 95% para os valores preditos pelos modelos ARIMA correspondentes a cada coluna do *data set*. Caso o valor observado estivesse fora desse intervalo para qualquer uma das colunas, foi considerado que houve uma falta.

Para a comparação com ARIMA tem-se as hipóteses:

$$\begin{cases} H_0 : F_{\text{Abordagem}} \leq F_{\text{ARIMA}} \\ H_a : F_{\text{Abordagem}} > F_{\text{ARIMA}} \end{cases}$$

As hipóteses constituem um teste monocaudal direito. O teste foi realizado com nível de significância $\alpha = 0,01$. Como o tamanho amostral n do experimento é igual a 30, foi utilizada a estatística teste padronizada z e σ aproximado como o desvio padrão amostral s.

Os resultados estão listados na Tabela 5.10.

Tabela 5.10: Resultados Experimento *Lighttpd*.

Medida	**Valor**
Média F-Measure ARIMA	0,2010910
Média F-Measure Abordagem	0,6616754
Desvio Padrão Abordagem	0,2759962
Desvio Padrão ARIMA	0,0992052
Tamanho Amostral (n)	30

Substituindo os valores observados na Equação 5.8:

$$z = \frac{0,6616754 - 0,2010910}{0,2759962/\sqrt{30}} \simeq 9,140436$$

Tem-se que a área sob a curva à direita do valor de z encontrado é aproximadamente zero. Logo, é possível rejeitar a hipótese nula H_0 com um *p-value* $\simeq 0$ e $\alpha = 1\%$, e concluir que há evidências suficientes de que o F-measure obtido pela abordagem é maior do que o alcançado com o uso de ARIMA.

Esse experimento mostrou que os resultados obtidos pela abordagem proposta foram significativamente melhores dos que os obtidos por SVM e ARIMA, o que recomenda seu uso ao invés dessas técnicas.

5.6 Considerações finais

Este capítulo apresentou diversos experimentos realizados com a abordagem proposta para detecção de faltas. Foram analisados dados gerados sinteticamente, bem como conjuntos de dados amplamente empregados na literatura para realizar uma avaliação inicial da abordagem. Posteriormente,

realizaram-se experimentos com processos reais, por meio da injeção de faltas via interceptação de chamadas de sistema. Análises estatísticas mostraram a um nível de erro $\alpha = 1\%$ que a abordagem obteve um *F-Measure* maior do que duas vezes o de SVM e ARIMA, técnicas que têm sido adotadas pelas áreas de Aprendizado de Máquina e Estatística, bem como por trabalhos relacionados (Zou e Liu, 2002; Hoffmann et al., 2007; Xue et al., 2007). No capítulo seguinte é apresentado um estudo complementar realizado sobre abordagens para predição de faltas.

CAPÍTULO

6

Predição de faltas

6.1 Considerações iniciais

Após atingir os objetivos do trabalho realizou-se um estudo complementar sobre predição de faltas. O comportamento não linear de processos, observado durante os experimentos realizados com injeção de faltas, motivou o estudo da predição de faltas por meio de sua modelagem com ferramentas de sistemas dinâmicos. Neste capítulo é apresentado esse estudo.

6.2 Sistemas Dinâmicos

De acordo com Alligood et al. (1997), um sistema dinâmico consiste de um conjunto de estados, juntamente com uma regra que determina o estado presente em termos de estados passados. Por exemplo, considere que a regra $f(x) = 2^x$ representa o número de bactérias observado a cada período de 1 hora de observação. Se a população inicial for $x = 5$, então após 1 hora haverá um total de $f(5) = 32$ bactérias. Após 2 horas haverá $f(f(5)) = 4.294.967.296$ bactérias. Esse cenário consiste em um sistema dinâmico em que os estados são níveis populacionais e a regra que o governa é $x_n = f(x_{n-1}) = 2^{x_{n-1}}$, sendo n o número de horas passadas e x_n o tamanho da população no instante n.

Um sistema dinâmico determinístico é aquele em que é possível determinar o estado presente apenas a partir de seus estados passados. Um exemplo de sistema aleatório, estocástico, ou não determinístico, é o lançamento de uma moeda honesta, onde o estado presente não depende de estados passados.

Considere um sistema que modela a posição de um objeto ao longo do

tempo, conforme ilustrado pela Figura 6.1. É possível observar que a posição do objeto apresenta um comportamento periódico.

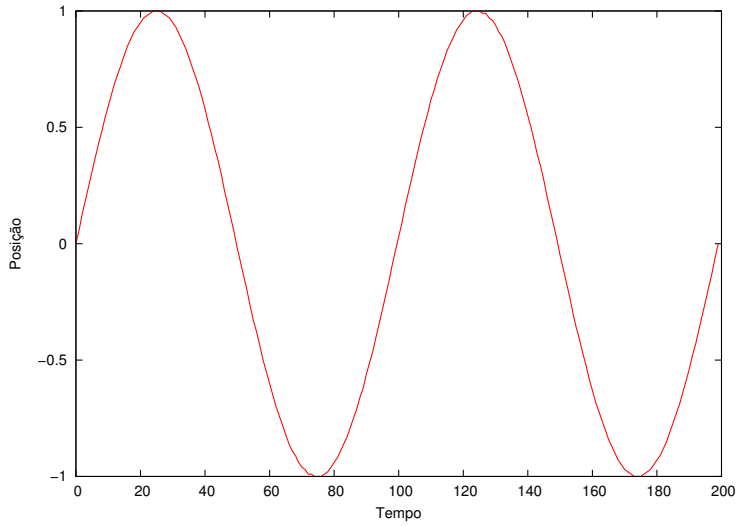


Figura 6.1: Posição de um objeto em função do tempo.

Um experimento interessante para compreender melhor a dinâmica desse sistema é fazer um gráfico de atraso (*delay plot*), no qual cada valor da série é plotado em função de valores passados, para um dado atraso T. Isso é chamado de reconstrução no espaço de coordenadas de atraso, ou gráfico de atraso (Alligood et al., 1997). A Figura 6.2 apresenta a reconstrução para o movimento periódico discutido, com atraso $T = 6$.

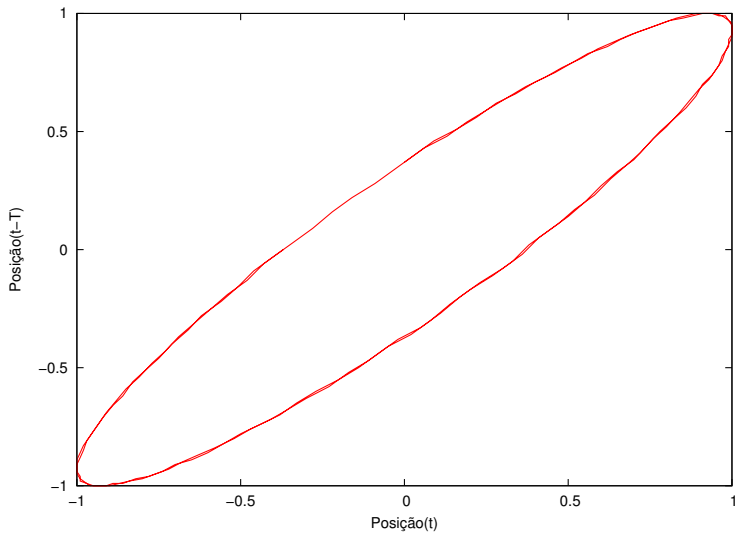
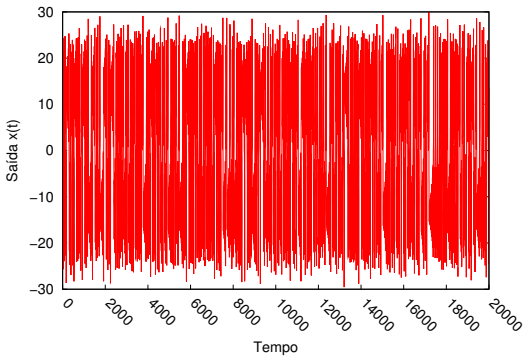


Figura 6.2: Reconstrução no espaço de coordenadas de atraso.

Essa análise permite observar que os estados do movimento periódico formam uma curva fechada simples. Analisar a dinâmica do sistema, com base em informações passadas, pode auxiliar a compreender seu comportamento, que poderia, a princípio, parecer aleatório.

Whitney (1936) estudou a reconstrução de sistemas e observou que essa técnica facilita a compreensão de comportamentos não observáveis quando



(a) Série de Lorenz.

(b) Série reconstruída com $m = 3$ e $T = 5$.

Figura 6.3: Reconstrução da série de Lorenz.

inseridos em um espaço de poucas dimensões. Isso o motivou a propor seu teorema da imersão, pelo qual é possível reconstruir uma série d-dimensional em um espaço $2d + 1$-dimensional, revelando regularidades anteriormente escondidas.

Takens (1981), baseado na teoria desenvolvida por Whitney, propôs o seu teorema da imersão, o qual visa reconstruir a dinâmica de um sistema utilizando deslocamentos no tempo, ao invés de mapear os estados do sistema para $2d + 1$ dimensões. Assim, é possível reconstruir uma série $x_1, x_2, \ldots, x_n$ em um espaço multidimensional $x_n(m, T) = (x_n, x_{n+T}, \ldots, x_{n+(m-1)T})$, o qual é denominado de espaço de coordenadas de atraso e dá-se o nome de dimensão embutida para m e dimensão de separação (*time delay*) para T. Dessa forma, é possível mapear uma série unidimensional, obtida a partir da regra de um sistema dinâmico, em pontos de um espaço m-dimensional.

Considere a Figura 6.3(a), que ilustra saídas da série do atrator de Lorenz (Lorenz, 1963). A reconstrução dessa série com dimensão embutida $m = 3$ e dimensão de separação $T = 5$ é apresentada na Figura 6.3(b). É possível observar que há uma regularidade no comportamento da série, anteriormente obscura em sua forma original. O desdobramento do comportamento no espaço de coordenadas de atraso permite compreender melhor como valores futuros da série estão atrelados com valores passados.

Uma questão importante para desdobrar completamente o comportamento de uma série é a escolha da dimensão embutida m e de separação T. Para séries lineares, é possível determinar a dimensão de separação por meio do uso de uma função de autocorrelação (Equação 6.1) (Abarbanel et al., 1993), em que T é o atraso (deslocamento temporal), $E[\cdot]$ é a esperança, μ a média da série e σ seu desvio padrão.

$$ACF(T) = \frac{E[(x_i - \mu)(x_{i+T} - \mu)]}{\sigma^2} \tag{6.1}$$

A obtenção da dimensão de separação pode feita pela técnica de Auto Informação Mútua (Fraser e Swinney, 1986). A Equação 6.2 apresenta o cálculo da Auto Informação Mútua, em que X e Y são variáveis distribuídas de acordo com $P_X(x)$ e $P_Y(y)$ e probabilidade conjunta $P_{XY(x,y)}$. Para obter a dimensão correta a técnica é aplicada com diferentes valores para o deslocamento T. Fraser e Swinney (1986) sugerem como escolha da dimensão de separação aquela para o qual se obtém o primeiro mínimo local.

$$I(X,Y) = \int P_{XY(x,y)} \log_2 \frac{P_{XY(x,y)}}{P_X(x)P_Y(y)} dxdy \tag{6.2}$$

A Figura 6.4 apresenta os valores de Auto Informação Mútua calculados para diversos deslocamentos temporais T. É possível observar que o primeiro mínimo local é 5, sendo portanto um valor adequado para a escolha da dimensão de separação (de Mello, 2009).

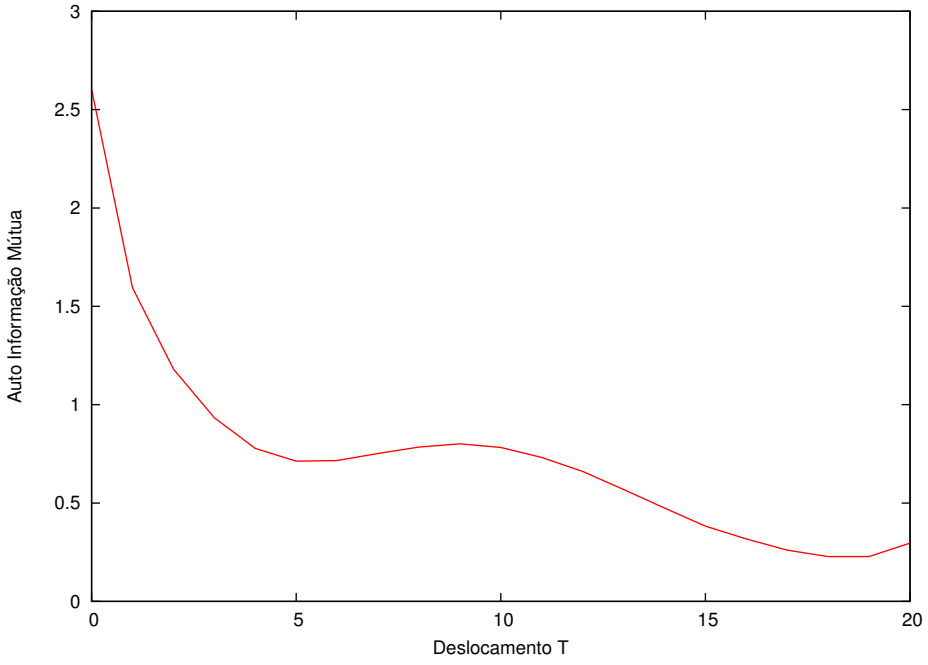


Figura 6.4: Auto Informação Mútua para série de Lorenz com diferentes dimensões de separação T.

Além da dimensão de separação é necessário determinar a dimensão embutida. Para isso é possível utilizar a técnica de Falsos Vizinhos mais Próximos (*False Nearest Neighbors*), proposta por Kennel et al. (1992). A técnica funciona encontrando os vizinhos mais próximos de cada ponto no espaço de coordenadas de atraso. O valor da dimensão embutida é incrementado, recalculando os vizinhos de cada ponto. Caso a distância para os vizinhos previamente identificados aumente, então significa que eles são “falsos vizinhos”,

fazendo-se necessários um número maior de dimensões para desdobrar completamente o comportamento da série. A distância entre os vizinhos é dada segundo a Equação 6.3, em que d é a dimensão embutida, x_n é um ponto da série no instante n e $x^{(i)}$ é o i-ésimo vizinho do ponto x_n. A cada iteração a técnica incrementa d e computa a variação obtida no cálculo da distância. Caso essa variação seja maior do que um limiar, então se assume que os pontos são falsos vizinhos. Kennel et al. (1992) indica que um limiar maior ou igual a 10 costuma fornecer bons resultados.

$$R_d^2(n,i) = \sum_{k=0}^{d-1} (x_{n+kT} - x_{n+kT}^{(i)})^2 \tag{6.3}$$

Aplicando a técnica de Falsos Vizinhos mais Próximos nos dados da série de Lorenz com dimensão de separação $T = 5$, obtém-se o gráfico ilustrado na Figura 6.5.

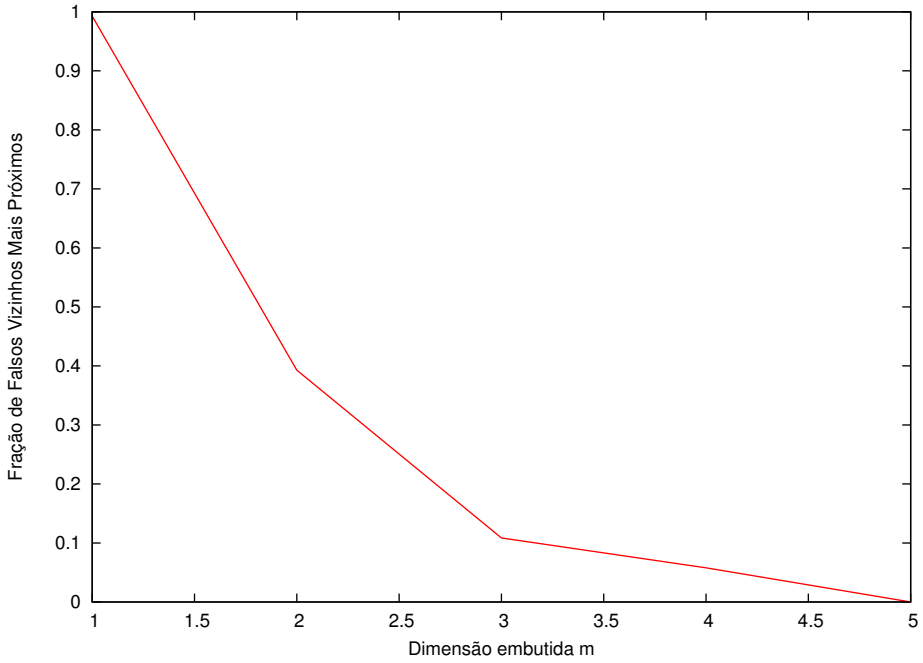


Figura 6.5: Fração de falsos vizinhos mais próximos para a série de Lorenz com diferentes valores para a dimensão embutida m.

É possível observar que um valor de 3 já fornece um bom índice de Falsos Vizinhos mais Próximos para o conjunto de dados da série de Lorenz (de Mello, 2009).

Após a obtenção de ambas dimensões de separação (T) e embutida (m) é possível aplicar o teorema de Takens e reconstruir a série no espaço de coordenadas de atraso, desdobrando seu comportamento, conforme ilustrado

anteriormente. Neste trabalho essa teoria foi utilizada com o intuito de desdobrar o comportamento de séries geradas a partir da observação de processos executados na presença de faltas.

6.3 Emprego de Sistemas Dinâmicos para predição de faltas

Como um ensaio inicial para o estudo de predição de faltas utilizou-se séries geradas a partir de execuções do utilitário `rsync`[1] v3.0.7, disponível para sistemas UNIX, e que permite sincronizar arquivos entre máquinas remotas, fornecendo um meio rápido para transferência incremental. Ele foi escolhido por ser livremente distribuído (licença GNU GPL) e amplamente utilizado (Sayood, 2003; Douglis e Davison, 2009). Além disso, o utilitário realiza diversos tipos de operações, tais como cópia de dados, compressão e comunicação via rede.

O primeiro passo foi executar o utilitário 30 vezes, copiando uma pasta contendo o código fonte da GNU Scientific Library [2] v1.14 para outra máquina na mesma rede local da máquina em que o processo `rsync` foi executado. O tempo médio entre faltas (MTBF) foi simulado seguindo uma distribuição Weibull com parâmetros $\lambda = 1.0$ e $k = 1.5$. O parâmetro k, ao ser configurado com um valor maior do que zero, simula um aumento na taxa de ocorrência de faltas conforme o tempo passa. Isso é ilustrado no histograma da Figura 6.6, em que é possível observar maior frequência para intervalos de tempo menores. A distribuição Weibull foi utilizada porque estudos têm mostrado que essa distribuição modela melhor a disponibilidade de sistemas distribuídos assíncronos, por sua característica *non-memoryless* (Nurmi et al., 2005). A Tabela 6.1 apresenta a proporção entre chamadas de sistema normais e chamadas em que faltas foram inseridas, considerando as 30 execuções.

A injeção de faltas se deu por meio da chamada de sistema `ptrace`, inserindo valores espúrios nos registradores, quando do retorno de chamadas de sistema de leitura.

Tabela 6.1: Proporção entre chamadas de sistema normais e chamadas em que faltas foram inseridas, considerando os 30 *traces* do `rsync`.

——	**Normal**	**Faltas**
Média	5306 (79%)	1341 (21%)
Desvio Padrão	13.6	7.72

[1] A ferramenta pode ser obtida em http://www.samba.org/rsync/. Último acesso: 24 de julho de 2010.

[2] Disponível em: ftp://ftp.gnu.org/gnu/gsl/. Último acesso: 09 de agosto de 2010.

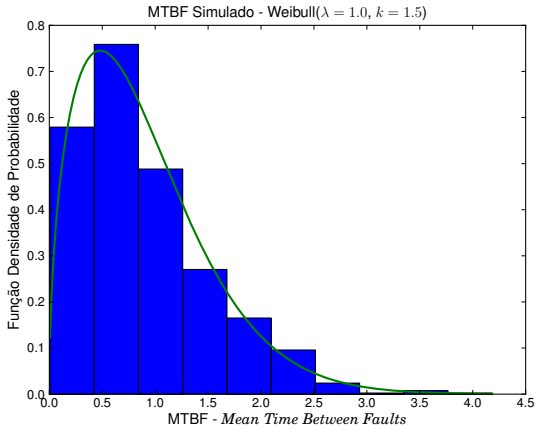


Figura 6.6: Histograma do tempo médio entre faltas simulado.

As informações capturadas sobre o processo consistiram de dados dos registradores e informações da ferramenta `vmstat`[3], tais como memória física disponível, número de chaveamentos de contexto por segundo e número de *swap* de blocos para disco por segundo. Esse conjunto de dados obtido é denominado *trace* a partir desse momento.

Após a obtenção dos *traces* do utilitário, consistindo dos dados capturados e da informação de injeção ou não de faltas (supervisão), aplicou-se a técnica de *Principal Component Analysis* (PCA) (Witten e Frank, 2005), a fim de obter uma projeção do conjunto de dados em apenas uma dimensão. Esse passo foi realizado para que as técnicas de sistemas dinâmicos, discutidas na seção anterior, pudessem ser utilizadas. A vantagem da técnica de PCA é que ela preserva a maior variabilidade dos dados na primeira dimensão, o que ocasiona menor perda de informação ao realizar a projeção. A Figura 6.7 apresenta a projeção do primeiro *trace* do `rsync`.

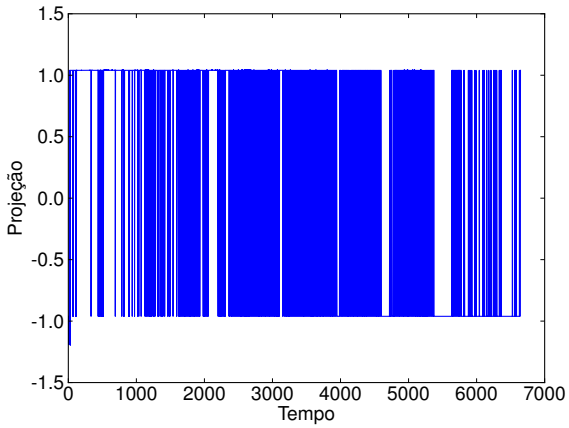


Figura 6.7: Projeção do *trace* obtido da primeira execução do `rsync`.

O passo seguinte à projeção dos dados foi estimar a dimensão de separação

[3]A lista completa de dados fornecidos pela ferramenta `vmstat` está disponível em http://linux.die.net/man/8/vmstat. Último acesso: 09 de agosto de 2010.

T. Para isso foi empregada a técnica de Auto Informação Mútua, conforme discutido na Seção 6.2. A Figura 6.8 apresenta a estimativa da Auto Informação Mútua para o *trace* 1. A ferramenta Tisean (Hegger et al., 1999) foi utilizada para estimar as dimensões de separação e embutida. Fraser e Swinney (1986) sugerem que o primeiro mínimo local seja escolhido como dimensão de separação, assim optou-se por utilizar $T = 2$.

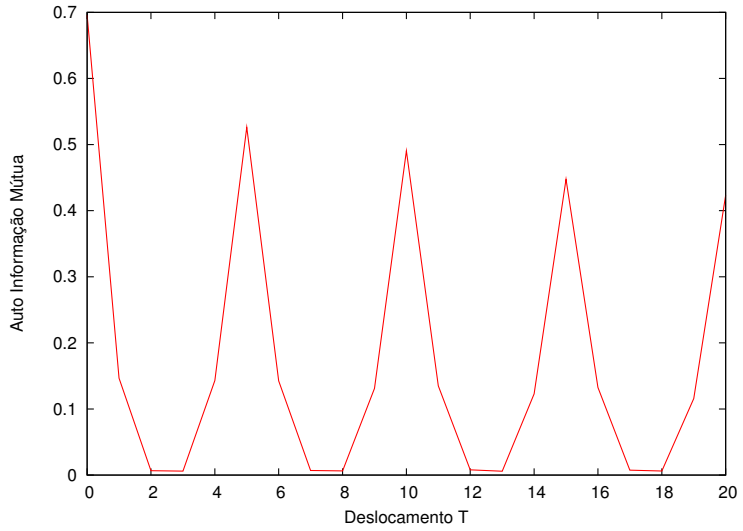


Figura 6.8: Auto Informação Mútua para o *trace* 1 com diferentes valores para a dimensão de separação T.

Após a determinação da dimensão de separação estimou-se a dimensão embutida m. Com esse intuito foi empregada a técnica de *False Nearest Neighbors*, discutida na Seção 6.2. A Figura 6.9 apresenta a fração de falsos vizinhos para diferentes valores de m.

Pelo gráfico da Figura 6.9 é possível observar que a menor fração de falsos vizinhos ocorre para a dimensão embutida $m = 4$. Para fins de visualização a reconstrução foi feita em 3 dimensões. Porém, a análise realizada de predição foi feita para $m = 4$.

A Figura 6.10 apresenta a reconstrução do *trace* 1 no espaço de coordenadas de atraso. É possível observar que houve uma separação entre pontos que pertenciam a ocorrência de faltas e os que pertenciam ao comportamento normal da aplicação. Dessa forma, o desdobramento no espaço de coordenadas de atraso forneceu um meio simplificado para o estudo do comportamento do utilitário na presença de faltas. O Apêndice A apresenta os gráficos dos 30 *traces* do `rsync`, juntamente com as reconstruções no espaço de coordenadas de atraso.

A partir da reconstrução no espaço de coordenadas de atraso obteve-se uma separação mais clara das instâncias que pertenciam a pontos com faltas.

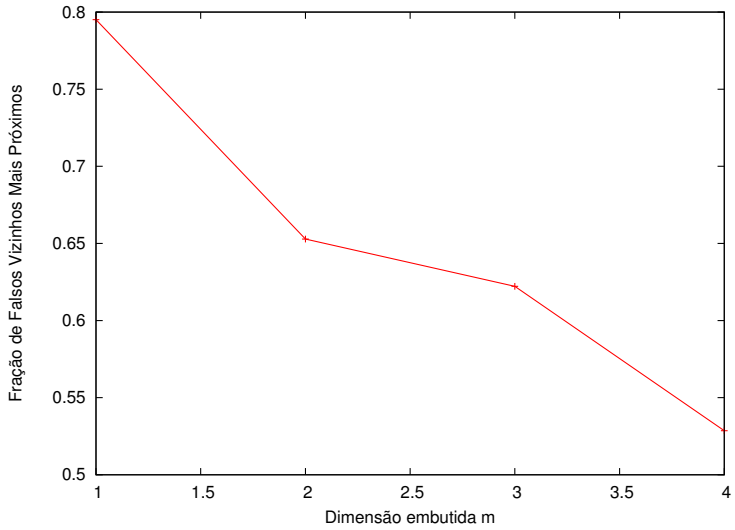


Figura 6.9: Fração de Falsos Vizinhos para o *trace* 1 com diferentes valores para dimensão embutida m.

O passo seguinte foi o desenvolvimento de um algoritmo para predição de faltas, utilizando a reconstrução obtida. Na seção seguinte as abordagens preditivas avaliadas são discutidas.

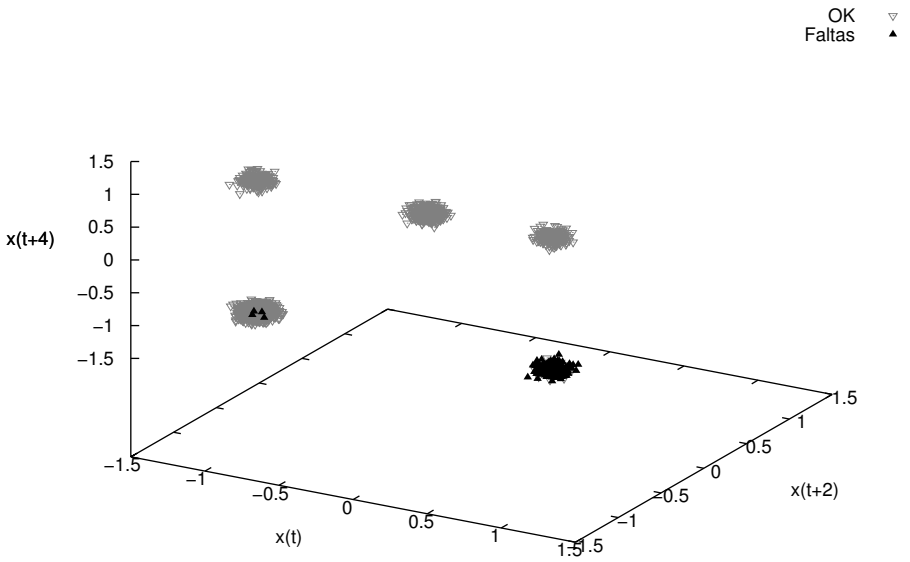


Figura 6.10: Reconstrução do *trace* 1 com $m = 3$ e $T = 2$. Os pontos destacados como faltas correspondem a faltas que ocorreram no ponto que compõe a quarta dimensão de cada instância no espaço de coordenadas de atraso com $m = 4$.

6.4 Abordagem preditiva baseada em janelas

A abordagem preditiva, inicialmente proposta, foi avaliada considerando uma janela de tempo para predizer faltas no futuro. Faltas que ocorram dentro

de uma janela indicam que haverá uma falta no futuro.

É importante destacar que essa abordagem, inicialmente avaliada, não considerou a reconstrução dos *traces* no espaço de coordenadas de atraso. Isso foi feito para mensurar, posteriormente, o impacto que técnicas de Sistemas Dinâmicos teriam na eficácia da mesma abordagem preditiva. A abordagem é listada no Algoritmo 3 e funciona como descrito a seguir.

Inicialmente, divide-se os dados em dois conjuntos, o primeiro $\mathcal{T}_{treino}$ para o treinamento, e o segundo $\mathcal{T}_{teste}$ para teste. O conjunto $\mathcal{T}_{treino}$ é submetido a técnica *kmeans* (Algoritmo 4) para obtenção de K centros representativos dos dados. O valor de K é fornecido pelo usuário. Após encontrar os centros, a rotina *top* (Algoritmo 5) seleciona C centros dos K encontrados, sendo esses C os que possuem o maior número de pontos com faltas. Esses centros são adicionados ao conjunto $\mathcal{F}$. Define-se $p : \mathbb{Z} \rightarrow \{0, 1\}$ como a função preditora, que mapeia instantes $t \in \mathbb{Z}$ para um valor no conjunto $\{0, 1\}$, sendo que 0 representa comportamento normal e 1 representa uma falta.

Durante a fase de teste, para uma dada janela de pontos no intervalo de i a $i + (m-2)T$, realiza-se a predição da seguinte forma: computa-se a distância Euclidiana (Algoritmo 6) do padrão j em relação ao centro de cada um dos grupos $G \in \mathcal{G}$, encontrados pela rotina *kmeans*. O centro do grupo c que minimiza a distância é selecionado para agrupar o padrão j. Caso c esteja no conjunto $\mathcal{F}$ para qualquer um dos elementos j na janela considerada, então se faz a predição de que no instante $t = i + (m-1)T$ ocorrerá uma falta. Caso todo $c \notin \mathcal{F}$ na janela, então se prediz que o comportamento no instante t será normal.

Os resultados obtidos estão listados na Tabela 6.2 e a ROC é mostrada na Figura 6.11. É possível observar que a técnica inicialmente considerada teve uma alta taxa de falsos positivos e baixa acurácia. Isso se deve ao fato de que essa abordagem inicial considera os pontos de uma janela de maneira isolada, ou seja, caso ocorra uma falta em qualquer instante de uma janela, prediz-se que ocorrerá uma falta na próxima janela. Na seção seguinte discute-se a proposta de melhoria para essa abordagem, que se baseia em Sistemas Dinâmicos, para explicitar as relações temporais entre os estados do processo e melhor modelá-los.

Tabela 6.2: Resultados da abordagem preditiva baseada em janelas para os 30 *traces* do `rsync` com parâmetros $K = 3$ e $C = 2$.

Medida	Acurácia	Precisão	Revocação	T. Falsos Positivos	T. Falsos Negativos
Média	0,28	0,11	0,98	0,78	0,02
Desvio Padrão	0,25	0,04	0,01	0,28	0,01

Algoritmo 3: Abordagem preditiva baseada em janelas

```
K e C são definidos pelo usuário.
T ← estimado via Auto Mutual Information;
m ← estimado via False Nearest Neighbors;
T_treino ← 2/3 dados;
T_teste ← 1/3 dados;
G ← kmeans(T_treino, K);
F ← top(G, C);
p : Z → {0, 1};
for I_i ∈ T_teste do
    janela ← [I_i : I_{i+(m−2)T}];
    predFault ← 0;
    for j ∈ janela do
        minDist ← ∞;
        c ← ∅;
        for G ∈ G do
            d ← euclid(j, μ_G);
            if d < minDist then
                minDist ← d;
                c ← G;
            end
        end
        if c ∈ F then
            predFault ← 1;
            break;
        end
    end
    if predFault == 1 then
        p(i + (m − 1)T) ← 1;
    end
    else
        p(i + (m − 1)T) ← 0;
    end
end
```

Algoritmo 4: Kmeans

- K é recebido como parâmetro (definido pelo usuário);
- Particione os padrões pertencentes a $\mathcal{T}_{treino}$ em $\mathcal{G} = \{G_1, G_2, \ldots, G_K\}$ conjuntos, de forma a minimizar a soma da distância quadrada intra cluster:
- $argmin(\mathcal{G}) \sum_{i=1}^{K} \sum_{x_j \in G_i} ||x_j - \mu_i||^2$, onde K é o número de grupos, x_j é o j-ésimo elemento do grupo G_i e μ_i é o centro do grupo G_i;
- Retorne $\mathcal{G}$.

Algoritmo 5: Top

- C é recebido como parâmetro (definido pelo usuário);
- $\mathcal{F} \leftarrow \{\}$;
- Para 1 até C, adicione a $\mathcal{F} : argmax(G)|G_F|$, em que G_F é o subconjunto de elementos com faltas em $G \in \mathcal{G} - \mathcal{F}$;
- Retorne $\mathcal{F}$.

Algoritmo 6: Euclid

- Retorne $\sqrt{\sum(j_{ik} - \mu_{Gk})^2}$, em que j_i é o i-ésimo padrão na janela considerada, μ_G é o centro do grupo G e k representa o k-ésimo atributo.

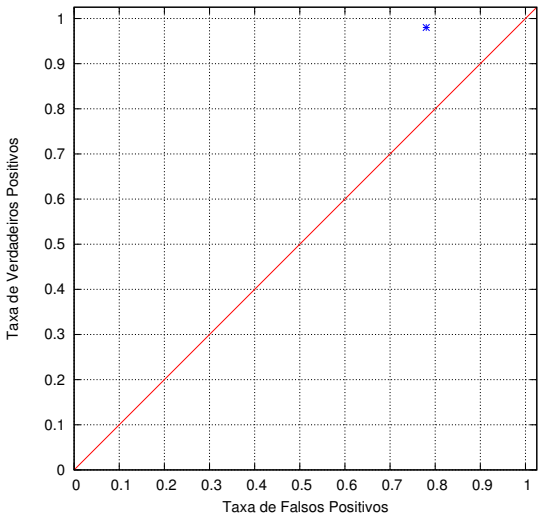


Figura 6.11: ROC para a abordagem preditiva baseada em janelas. O ponto no gráfico representa o classificador obtido. A reta indica o resultado esperado de um classificador aleatório. Pontos no canto superior esquerdo representam melhores resultados.

6.5 Abordagem preditiva proposta

Conforme discutido na seção anterior, considerar isoladamente pontos em uma janela temporal para prever se haverá ou não faltas no futuro não leva a resultados satisfatórios. Nesse sentido, uma nova abordagem foi proposta. Nessa versão considera-se os mesmos dados que a anterior, porém reconstruídos no espaço de coordenadas de atraso. Conforme mostrado anteriormente (Figura 6.10), nesse espaço torna-se mais simples a diferenciação entre comportamento normal e com faltas.

A abordagem proposta, listada no Algoritmo 7, funciona de maneira similar à anterior, porém ao invés de considerar janelas temporais, considera-se tuplas no espaço de coordenadas de atraso. Inicialmente, divide-se os dados em dois conjuntos, o primeiro $\mathcal{T}_{treino}$ para o treinamento, e o segundo $\mathcal{T}_{teste}$ para teste. O conjunto $\mathcal{T}_{treino}$ é submetido a técnica *kmeans* (Algoritmo 4) para obtenção de K centros representativos dos dados. O valor de K é fornecido pelo usuário. Após encontrar os centros, a rotina *top* (Algoritmo 5) seleciona C centros dos K encontrados, sendo esses C os que possuem o maior número de pontos com faltas. Esses centros são adicionados ao conjunto $\mathcal{F}$. Define-se $p : \mathbb{Z} \rightarrow \{0, 1\}$ como a função preditora, que mapeia instantes $t \in \mathbb{Z}$ para um valor no conjunto $\{0, 1\}$, sendo que 0 representa comportamento normal e 1 representa uma falta.

Durante a fase de teste, para a i-ésima entrada I, realiza-se a predição da seguinte forma. Computa-se a distância Euclidiana (Algoritmo 6) do padrão

I_i ao centro de cada um dos grupos $G \in \mathcal{G}$, encontrados pela rotina *kmeans*. O centro do grupo c que minimiza a distância é selecionado para agrupar o padrão I_i. Caso c esteja no conjunto $\mathcal{F}$, então se prediz uma falta no instante $t = i + (m-1)T$. Caso $c \notin \mathcal{F}$, então se prediz que o comportamento no instante t será normal.

Tabela 6.3: Resultados da abordagem preditiva para os 30 *traces* do `rsync` com parâmetros $K = 3$ e $C = 2$.

Medida	Acurácia	Precisão	Revocação	T. Falsos Positivos	T. Falsos Negativos
Média	0,80	0,36	0,97	0,21	0,03
Desvio Padrão	0,20	0,11	0,01	0,22	0,01

Algoritmo 7: Abordagem preditiva proposta

```
K e C são definidos pelo usuário.
T ← estimado via Auto Mutual Information;
m ← estimado via False Nearest Neighbors;
T_treino ← 2/3 dados;
T_teste ← 1/3 dados;
G ← kmeans(T_treino, K);
F ← top(G, C);
p : Z → {0, 1};
for I_i ∈ T_teste do
    minDist ← ∞;
    c ← ∅;
    for G ∈ G do
        d ← euclid(I_i, μ_G);
        if d < minDist then
            minDist ← d;
            c ← G;
        end
    end
    if c ∈ F then
        p(i + (m − 1)T) ← 1;
    end
    else
        p(i + (m − 1)T) ← 0;
    end
end
```

A aplicação do algoritmo proposto para predição forneceu os resultados listados na Tabela 6.3. É possível observar que a técnica atingiu um alto valor $(0,97)$ para a Revocação (Taxa de Verdadeiros Positivos) e um baixo índice de Falsos Positivos $(0,21)$ e Falsos Negativos $(0,03)$. A Figura 6.12 apresenta a ROC gerada para a abordagem proposta.

Com esses experimentos foi possível observar que o desdobramento do comportamento de uma aplicação no espaço de coordenadas de atraso pode facilitar substancialmente a predição correta de faltas, dado que a acurácia média da técnica foi de $0,28$ para $0,80$.

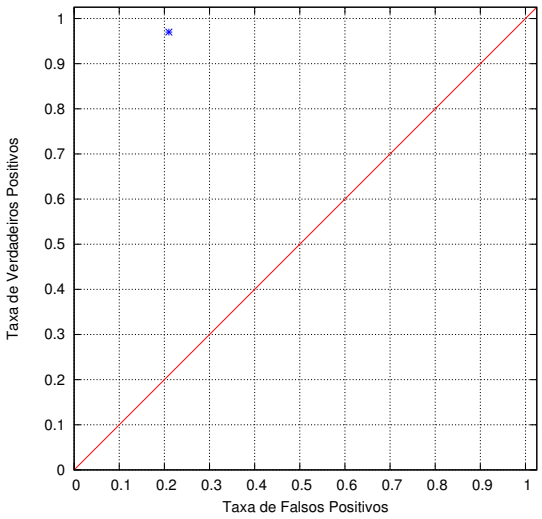


Figura 6.12: ROC para a abordagem preditiva proposta. O ponto no gráfico representa o classificador obtido. A reta indica o resultado esperado de um classificador aleatório. Pontos no canto superior esquerdo representam melhores resultados.

6.6 Avaliação de impacto da abordagem preditiva proposta na disponibilidade de sistemas

Salfner et al. (2005) propõem uma metodologia para avaliar o impacto de uma abordagem de predição de faltas na disponibilidade de um sistema. Essa metodologia foi utilizada para avaliar como a abordagem preditiva proposta afeta um sistema, em termos do número de horas em que ele se encontra disponível.

Inicialmente, define-se a disponibilidade ($\mathcal{D}$) como a razão entre $MTTF$ (*Mean Time To Fault*) e o total de $MTTF + MTTR$ (*Mean Time to Repair*), conforme a Equação 6.4 (Salfner et al., 2005).

$$\mathcal{D} = \frac{MTTF}{MTTF + MTTR} \tag{6.4}$$

Utiliza-se então as medidas de Precisão e Revocação como indicadores para avaliar como a abordagem preditiva aumenta o $MTTF$ do sistema. A precisão pode ser entendida como a razão entre o número de alarmes corretos e o número total de alarmes gerados. A revocação pode ser compreendida como a razão entre o número de alarmes corretos e o número de faltas ocorridas.

O reparo, bem como medidas preventivas iniciadas pela predição de faltas, afetam o $MTTR$ e $MTTF$. O efeito no $MTTR$ pode ser medido pelo fator de melhoramento de reparo r_f, dado um alarme correto a_c, conforme a Equação 6.5 (Salfner et al., 2005). $MTTR_{prep}$ é o $MTTR$ no caso de uma ação de reparo

ter sido preparada para uma falta predita. O fator r_f pode assumir valores no intervalo $[0, \infty)$. Um fator menor do que 1 representa uma melhoria, já que o tempo para reparo diminui.

$$r_f = \frac{MTTR_{prep}}{MTTR}/a_c \tag{6.5}$$

Já o efeito no $MTTF$ é medido segundo duas probabilidades: P_p (Equação 6.6), que indica a probabilidade de uma falta ser prevenida dado um alerta correto e P_e (Equação 6.7), que é a probabilidade de que um alarme cause uma falta extra (adicional), em função da ação de reparo tomada.

$$P_p = P(F_{prevenida}/a_c) \tag{6.6}$$

$$P_e = P(falta\ adicional) \tag{6.7}$$

Para um sistema que adota medidas de reparo guiadas por um algoritmo preditivo de faltas, define-se um tempo médio para faltas $MTTF'$ e um tempo médio para reparo $MTTR'$. O $MTTR'$ é definido como uma mistura de $MTTR$ e $MTTR_{prep}$ ponderados pelas probabilidades de (1 - Revocação) e Revocação, conforme a Equação 6.8.

$$MTTR' = (1 + recall \cdot (r_f - 1)) \cdot MTTR \tag{6.8}$$

Para computar o valor de $MTTF'$ é necessário medir o número de faltas ocorridas em um intervalo de tempo arbitrário δt de execução do sistema sem o mecanismo de predição de faltas ativo. Esse número de faltas f é então computado por $f = \delta t/(MTTF + MTTR)$. Ao aplicar um mecanismo preditivo de faltas, o número de faltas f é alterado de duas formas, com a prevenção de algumas faltas e a possível adição de outras (Equação 6.9). Os valores de $f_{prevenidas}$ e f_{extras} podem ser calculados como $P_p \cdot a_c$ e $P_e \cdot a$, respectivamente, sendo a_c o número de alertas corretos e a o número de alertas total. O $MTTF'$ pode então ser computado conforme a Equação 6.10 (Salfner et al., 2005).

$$f' = f - f_{prevenidas} + f_{extras} \tag{6.9}$$

$$MTTF' = \frac{\delta t}{f'} - MTTR' \tag{6.10}$$

Combinando as Equações 6.8 e 6.9 com 6.10 é possível reescrever $MTTF'$ conforme a Equação 6.11.

$$MTTF' = \frac{MTTF + MTTR}{1 - P_p \cdot recall + P_e \cdot \frac{recall}{precision}} - (1 + recall \cdot (r_f - 1)) \cdot MTTR \qquad (6.11)$$

As Equações 6.8 e 6.11 podem ser então combinadas para computar a disponibilidade $\mathcal{D}'$ de um sistema com reparos guiados por um algoritmo de predição de faltas, conforme a Equação 6.12, em que α é o termo computado segundo a Equação 6.13 e $MTTR$ e $MTTF$ são os tempos calculados para o sistema original (Salfner et al., 2005).

$$\mathcal{D}' = \mathcal{D} + \alpha \frac{MTTR}{MTTF + MTTR} \qquad (6.12)$$

$$\alpha = 1 - (1 + recall \cdot (r_f - 1)) \cdot \left(1 - P_p \cdot recall + P_e \cdot \frac{recall}{precision}\right) \qquad (6.13)$$

Considere um sistema arbitrário[4], que possui seis minutos (um décimo de hora) de *downtime* por 1000 horas de operação, o que corresponde a uma disponibilidade de "quatro noves", probabilidade preventiva $P_p = 0.9$, probabilidade de faltas extras $P_e = 0.1$ e fator de melhoramento de reparo $r_f = 0.5$. Os valores de Precisão e Revocação obtidos pela abordagem preditiva proposta resultam em uma disponibilidade $\mathcal{D}' \simeq 0.999992$. Isso representa aproximadamente 12500 horas de disponibilidade com 6 minutos de *downtime*, ou seja, uma ordem de magnitude maior disponibilidade do que o sistema original (999.9 horas), evidenciando o benefício da utilização da abordagem proposta.

A mesma análise, considerando esse mesmo sistema, mas utilizando a abordagem preditiva inicial (baseada em janelas), teria sua disponibilidade aumentada para aproximadamente 5000 horas com 6 minutos de *downtime*. Essa análise sugere que a abordagem preditiva proposta, baseada na reconstrução do comportamento do sistema no espaço de coordenadas de atraso, e agrupamento desse comportamento, pode levar a uma substancial melhoria na disponibilidade de um sistema.

6.7 Considerações finais

Neste capítulo foi apresentado um estudo complementar e preliminar sobre o emprego de Sistemas Dinâmicos para predição de faltas. A reconstrução do comportamento do sistema no espaço de coordenadas de atraso auxiliou a compreensão e a separação entre comportamento normal e com faltas, ocorridos durante a execução do utilitário empregado, o programa de sincronização

[4]A definição desse sistema segue a mesma especificação de valores utilizada em Salfner et al. (2005).

de arquivos `rsync`. A abordagem preditiva proposta apresentou bons resultados, podendo aumentar a disponibilidade do sistema em até uma ordem de magnitude. Foi também avaliada, inicialmente, uma abordagem preditiva baseada em janelas, entretanto ela não se mostrou tão eficaz quanto a baseada na reconstrução do comportamento, especialmente devido ao fato de considerar pontos isoladamente, e não sua composição no espaço de coordenadas de atraso. No capítulo seguinte são apresentadas as conclusões desta dissertação.

CAPÍTULO

7

Conclusões

Diversos trabalhos têm sido propostos na literatura da área de detecção faltas, os quais podem ser divididos em três grandes áreas. Os trabalhos que adotam abordagens baseadas em *heartbeats* apresentam a limitação de serem capazes de detectar somente falhas e não faltas, as quais são o motivo gerador de falhas. Os trabalhos baseados na teoria estatística de séries temporais geralmente assumem um comportamento linear do sistema, o que muitas vezes não é verdade. Técnicas baseadas em aprendizado de máquina visam superar ambas limitações, porém muitas vezes supõem que os dados são gerados de forma independente e identicamente distribuída (i.i.d.), o que também não é realidade para todos os casos.

Este trabalho apresentou uma nova abordagem para detecção de faltas, que leva em conta o comportamento de processos e visa superar as limitações encontradas nas outras técnicas propostas na literatura. A hipótese assumida é que informações sobre o histórico de um processo, tais como chamadas realizadas, argumentos passados e retornos de funções, são úteis para caracterizar seu comportamento ao longo do tempo.

Nesse contexto, foi proposta uma nova abordagem de detecção de faltas que busca modelar o comportamento de processos, no tempo, por meio de cadeias de Markov. Tais cadeias modelam estados representativos que um processo visita ao longo de sua execução, tais como “escrevendo dados”, “lendo dados”, “transmitindo dados via rede”, etc. Isso permite formalizar o comportamento normal de um processo.

A variação na Entropia dessas cadeias, ou seja, no nível de certeza de qual estado provavelmente será visitado, denota o nível de novidade sobre o comportamento do processo ao longo do tempo. Esse nível, medido entre

observações consecutivas, é a principal medida utilizada pela abordagem para detectar faltas.

Experimentos realizados mostraram, com evidências estatísticas a um nível de erro $\alpha = 1\%$, que a abordagem proposta pôde superar em duas vezes o *F-measure* obtido por técnicas como máquinas de vetores de suporte (SVM) e modelos auto-regressivos integrados de médias móveis (ARIMA).

Os bons resultados obtidos com a abordagem devem-se, principalmente, a dois fatores. O primeiro é não assumir que os dados foram gerados de maneira i.i.d., ou seja, de forma idêntica e de amostras independentes. Isso faz com que sejam consideradas as relações causais entre observações temporais dos processos analisados. A cadeia de Markov permite capturar esse relacionamento causal ao longo do tempo. Além disso, o segundo componente responsável pelos resultados é o fato de a abordagem não assumir modelos lineares, o que é conseguido pelo emprego de funções de base radial, que são aproximadores universais.

Finalmente, após cumprir o objetivo proposto neste trabalho, realizou-se um estudo complementar sobre o uso de técnicas de Sistemas Dinâmicos para predição de faltas. Para isso foi estudada a reconstrução do comportamento de processos no espaço de coordenadas de atraso, o que simplifica sua compreensão e modelagem.

A técnica preditiva proposta foi avaliada quanto ao aumento da disponibilidade de sistemas, e foi constatado que ela pode fornecer um aumento de até uma ordem de magnitude no número de horas que um sistema se encontra disponível. A abordagem foi comparada a uma outra, avaliada inicialmente, e que considerava janelas temporais para predizer faltas futuras. Entretanto, a abordagem proposta mostrou-se superior, pois considera a composição do comportamento do processo no espaço de coordenadas de atraso, enquanto a técnica avaliada inicialmente considera pontos de uma janela de forma isolada, o que faz com que relações importantes sejam perdidas na identificação do comportamento na presença de faltas.

Os resultados obtidos foram considerados promissores, sendo objeto das seguintes publicações:

- PEREIRA, C. M. M.; MELLO, R. F. Um Detector de Faltas Baseado no Comportamento de Processos. In: *Fórum de Pós-Graduação da I^{a}Escola Regional de Alto Desempenho de SP (ERAD-SP)*, São Paulo, Brasil: Sociedade Brasileira de Computação, p. 1–4, 2010.

- PEREIRA, C. M. M.; MELLO, R. F. A Radial Basis Function Neural Network Approach to Detect Novelties: Applications on Health Datasets. In: *The Second IEEE International Conference on Ubi-Media Computing*, Tamsui, Taiwan: IEEE Computer Society, p. 1–6, 2009.

- PEREIRA, C. M. M.; MELLO, R. F. Behavioral Study of UNIX Commands in a Faulty Environment. In: *The 8th International Conference on Dependable, Autonomic and Secure Computing*, Chengdu, China: IEEE Computer Society, p. 1–6, 2009.

E um artigo submetido para um periódico internacional:

- PEREIRA, C. M. M.; MELLO, R. F. Learning Process Behavior for Fault Detection. *International Journal on Artificial Intelligence Tools*, p. 1–28, 2010.

O autor recebeu uma menção honrosa no Fórum de Pós Graduação da Primeira Escola Regional de Alto Desempenho de São Paulo (ERAD-SP) pelo trabalho apresentado, o qual foi considerado um dos melhores.

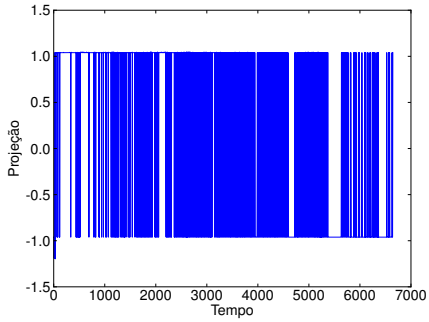
Trabalhos futuros podem dar continuidade à pesquisa realizada investigando a identificação de *root causes*, ou causas originais das faltas. Enquanto os estudos realizados nesta dissertação investigaram a detecção ou predição de faltas, não investigou-se a descoberta da causa (componente de *hardware* ou *software*) que originou a falta. Outra linha interessante é a criação de modelos baseados em regras, que facilitem a interpretação de um operador humano sobre as causas que levaram o sistema à ocorrência de uma falta.

APÊNDICE

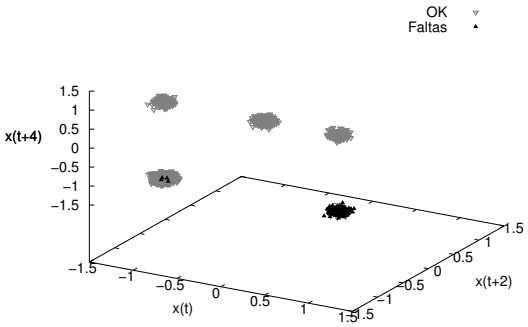
A

Reconstruções dos *traces* do utilitário `rsync`

Neste apêndice são apresentados os gráficos das projeções em uma dimensão dos 30 *traces* do utilitário `rsync`, obtidos por meio da técnica PCA e discutidos no Capítulo 6, bem como a reconstrução dos mesmos no espaço de coordenadas de atraso com dimensão embutida $m = 3$ e dimensão de separação $T = 2$. É possível observar que o comportamento com faltas tende a se agrupar em regiões bem definidas.

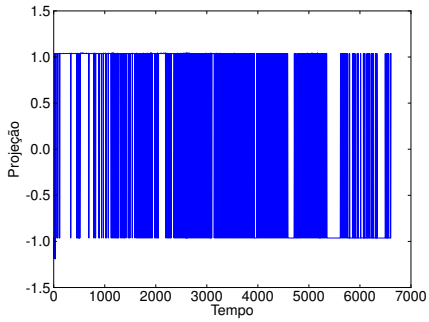


(a) Projeção do *trace* 1.

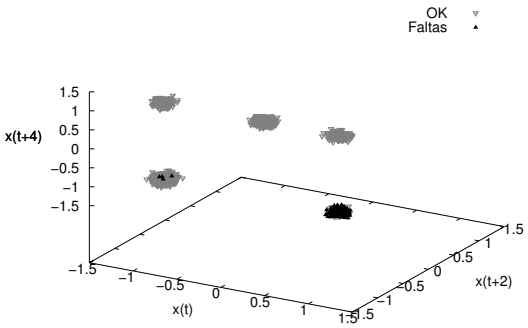


(b) Reconstrução do *trace* 1

Figura A.1: *Trace* 1 do utilitário rsync, juntamente com sua reconstrução.

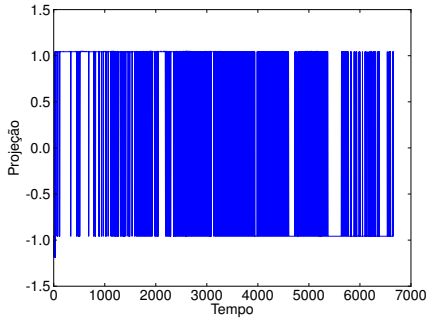


(a) Projeção do *trace* 2.

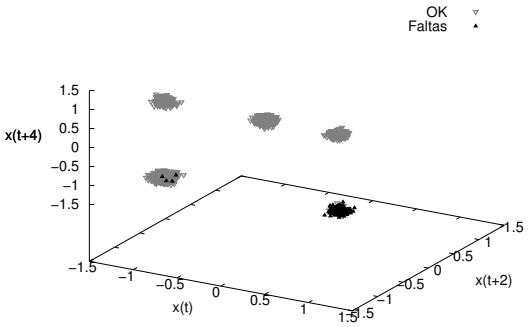


(b) Reconstrução do *trace* 2

Figura A.2: *Trace* 2 do utilitário rsync, juntamente com sua reconstrução.

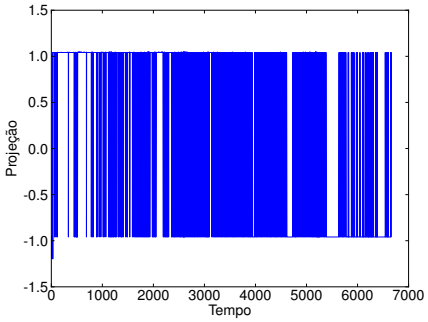


(a) Projeção do *trace* 3.

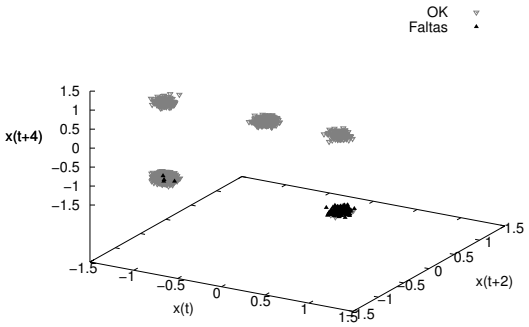


(b) Reconstrução do *trace* 3

Figura A.3: *Trace* 3 do utilitário rsync, juntamente com sua reconstrução.

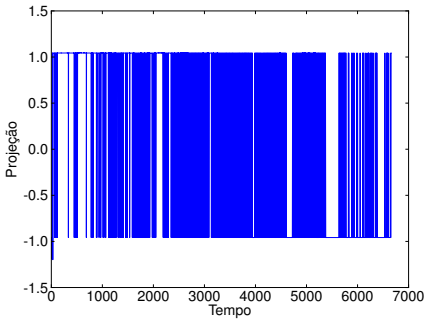


(a) Projeção do *trace* 4.

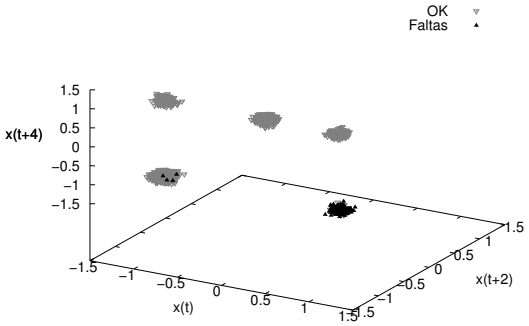


(b) Reconstrução do *trace* 4

Figura A.4: *Trace* 4 do utilitário rsync, juntamente com sua reconstrução.

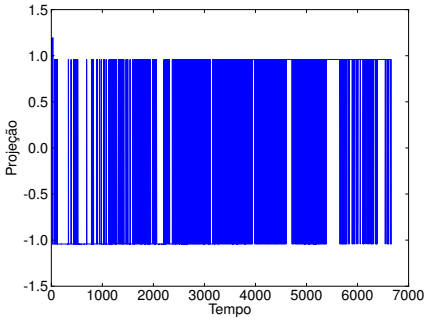


(a) Projeção do *trace* 5.

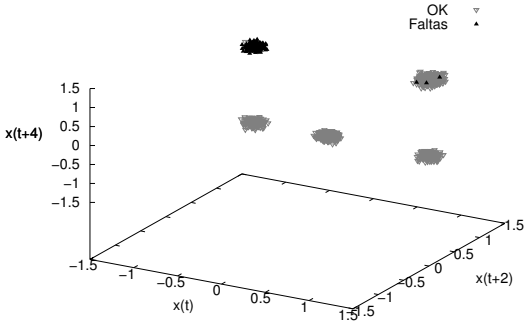


(b) Reconstrução do *trace* 5

Figura A.5: *Trace* 5 do utilitário rsync, juntamente com sua reconstrução.

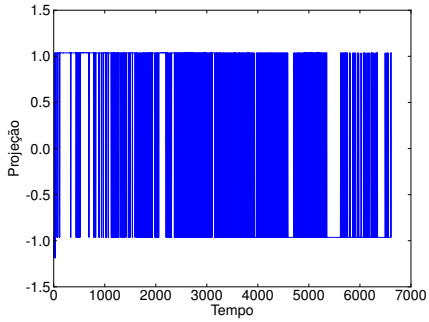


(a) Projeção do *trace* 6.

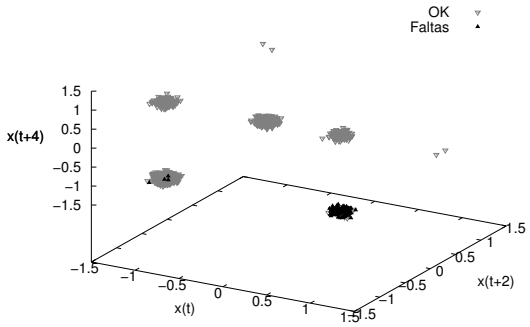


(b) Reconstrução do *trace* 6

Figura A.6: *Trace* 6 do utilitário rsync, juntamente com sua reconstrução.

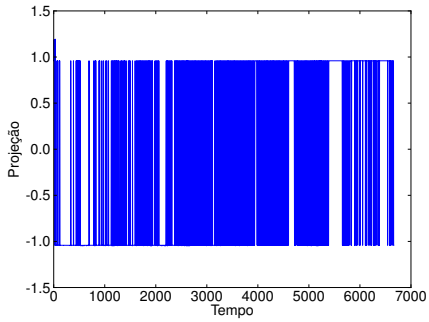


(a) Projeção do *trace* 7.

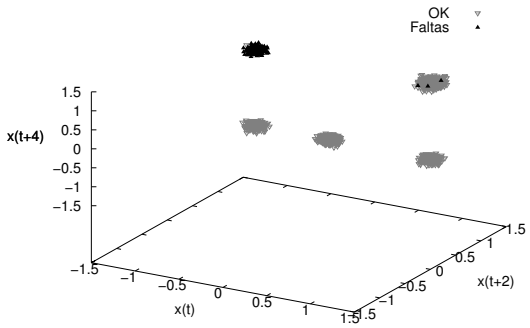


(b) Reconstrução do *trace* 7

Figura A.7: *Trace* 7 do utilitário rsync, juntamente com sua reconstrução.

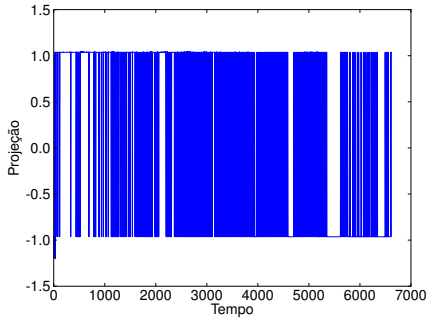


(a) Projeção do *trace* 8.

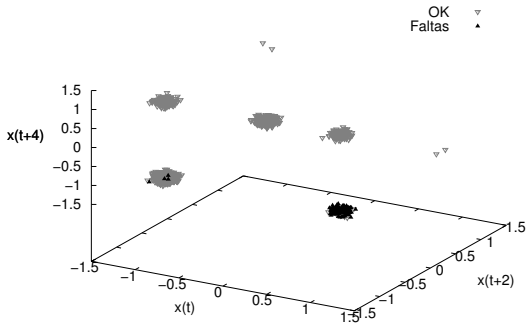


(b) Reconstrução do *trace* 8

Figura A.8: *Trace* 8 do utilitário rsync, juntamente com sua reconstrução.

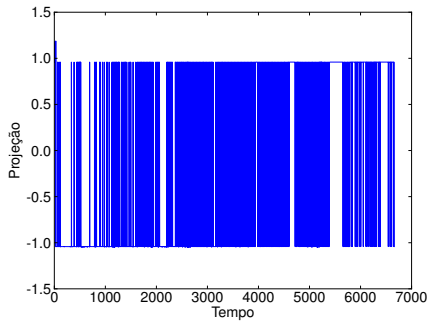


(a) Projeção do *trace* 9.

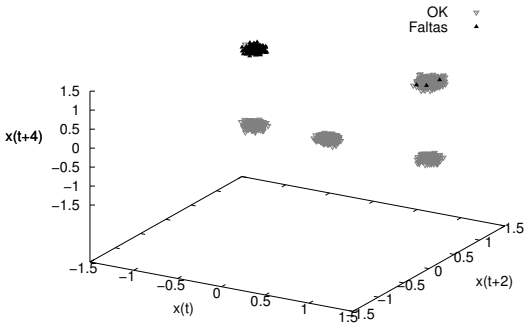


(b) Reconstrução do *trace* 9

Figura A.9: *Trace* 9 do utilitário rsync, juntamente com sua reconstrução.

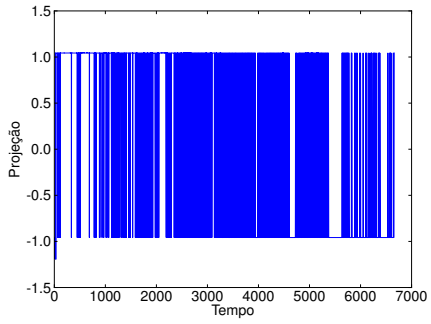


(a) Projeção do *trace* 10.

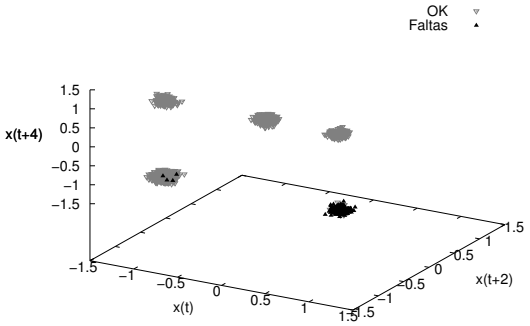


(b) Reconstrução do *trace* 10

Figura A.10: *Trace* 10 do utilitário rsync, juntamente com sua reconstrução.

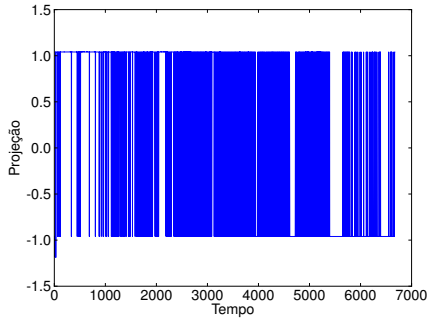


(a) Projeção do *trace* 11.

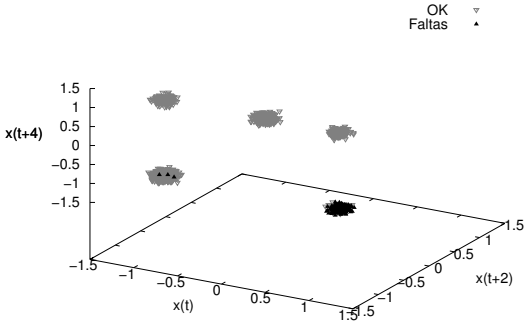


(b) Reconstrução do *trace* 11

Figura A.11: *Trace* 11 do utilitário rsync, juntamente com sua reconstrução.

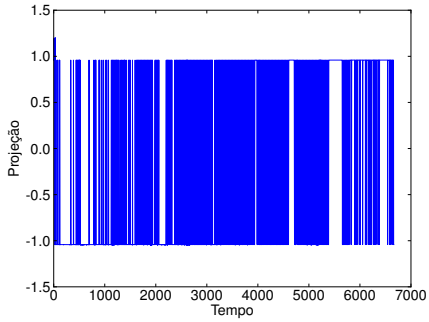


(a) Projeção do *trace* 12.

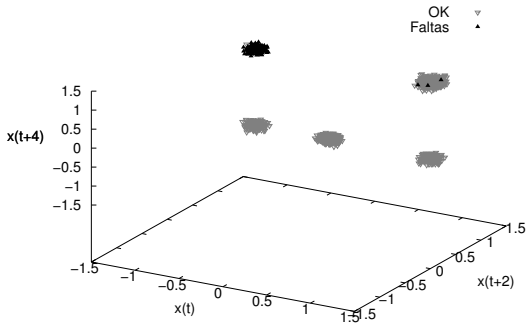


(b) Reconstrução do *trace* 12

Figura A.12: *Trace* 12 do utilitário rsync, juntamente com sua reconstrução.

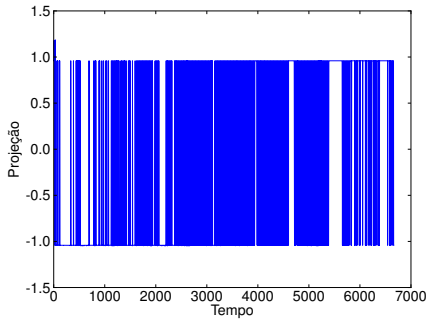


(a) Projeção do *trace* 13.

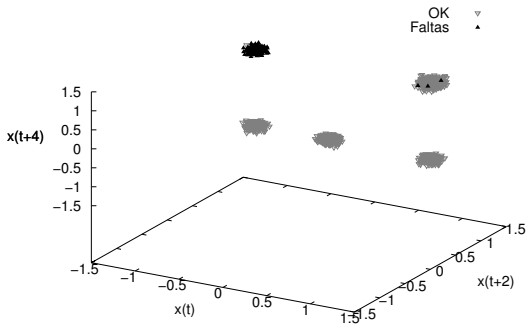


(b) Reconstrução do *trace* 13

Figura A.13: *Trace* 13 do utilitário rsync, juntamente com sua reconstrução.

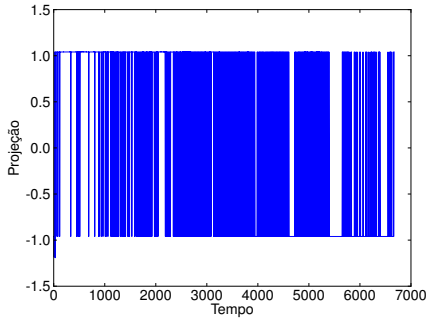


(a) Projeção do *trace* 14.

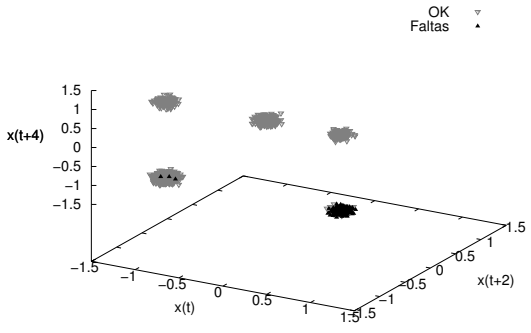


(b) Reconstrução do *trace* 14

Figura A.14: *Trace* 14 do utilitário rsync, juntamente com sua reconstrução.

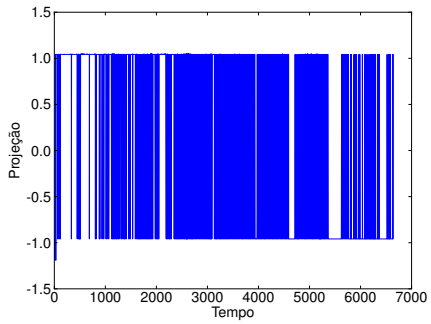


(a) Projeção do *trace* 15.

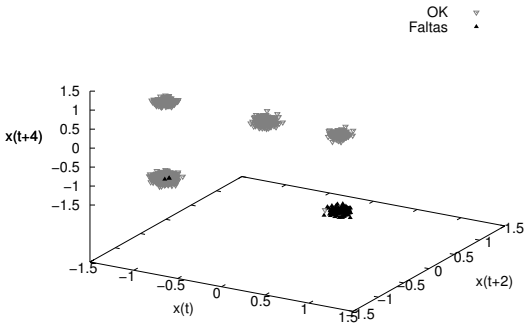


(b) Reconstrução do *trace* 15

Figura A.15: *Trace* 15 do utilitário rsync, juntamente com sua reconstrução.

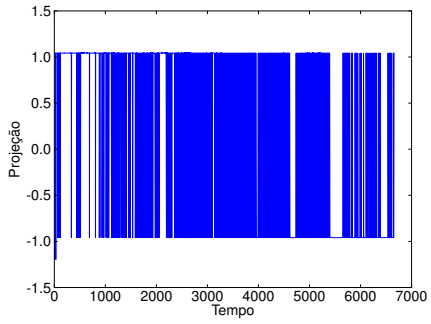


(a) Projeção do *trace* 16.

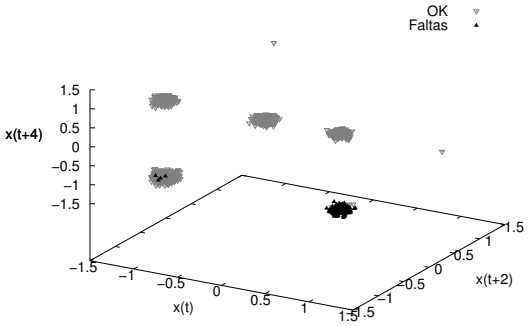


(b) Reconstrução do *trace* 16

Figura A.16: *Trace* 16 do utilitário rsync, juntamente com sua reconstrução.

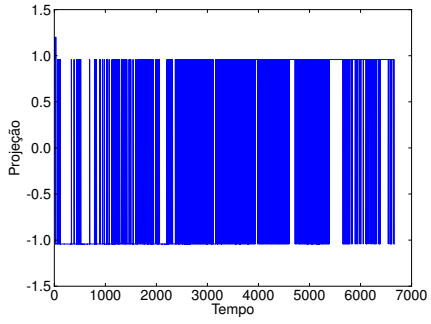


(a) Projeção do *trace* 17.

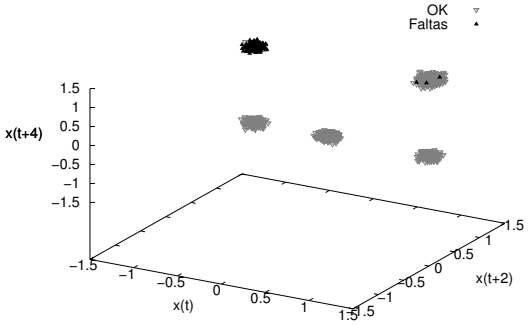


(b) Reconstrução do *trace* 17

Figura A.17: *Trace* 17 do utilitário rsync, juntamente com sua reconstrução.

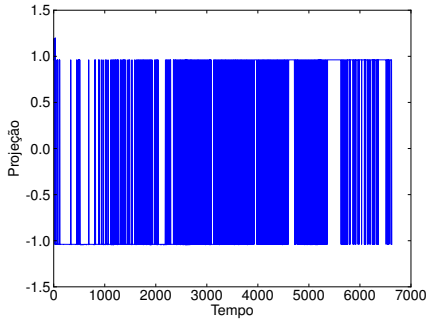


(a) Projeção do *trace* 18.

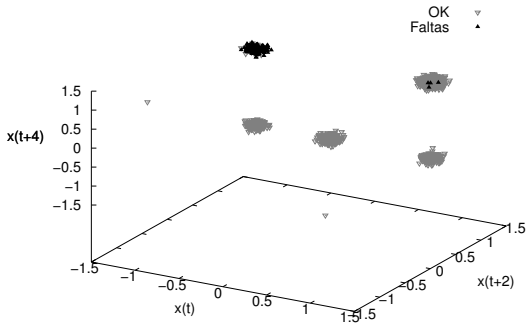


(b) Reconstrução do *trace* 18

Figura A.18: *Trace* 18 do utilitário rsync, juntamente com sua reconstrução.

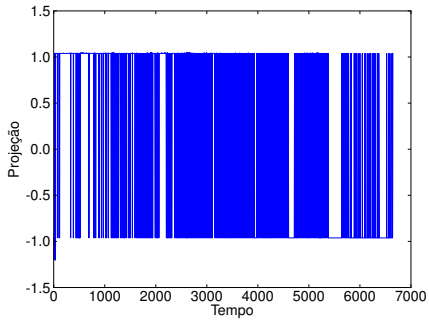


(a) Projeção do *trace* 19.

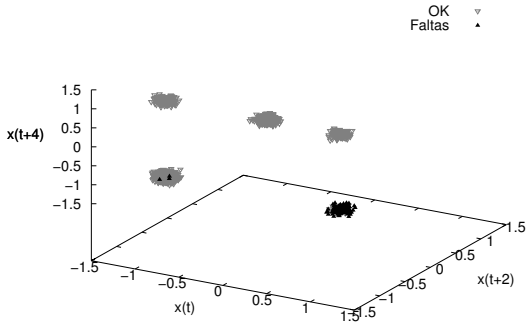


(b) Reconstrução do *trace* 19

Figura A.19: *Trace* 19 do utilitário rsync, juntamente com sua reconstrução.

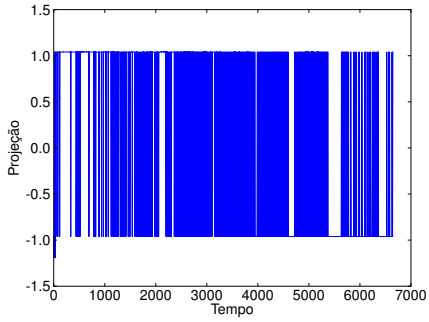


(a) Projeção do *trace* 20.

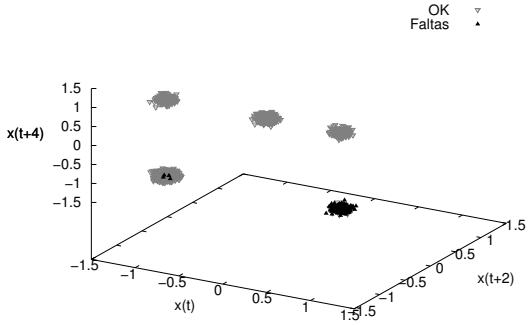


(b) Reconstrução do *trace* 20

Figura A.20: *Trace* 20 do utilitário rsync, juntamente com sua reconstrução.

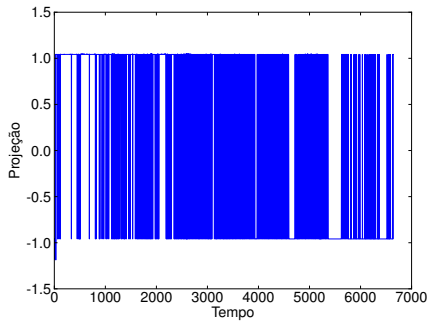


(a) Projeção do *trace* 21.

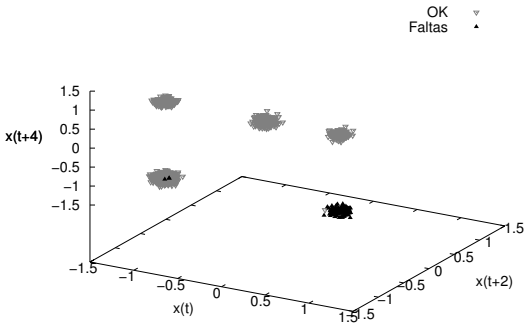


(b) Reconstrução do *trace* 21

Figura A.21: *Trace* 21 do utilitário rsync, juntamente com sua reconstrução.

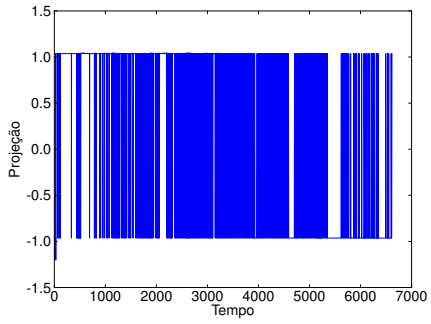


(a) Projeção do *trace* 22.

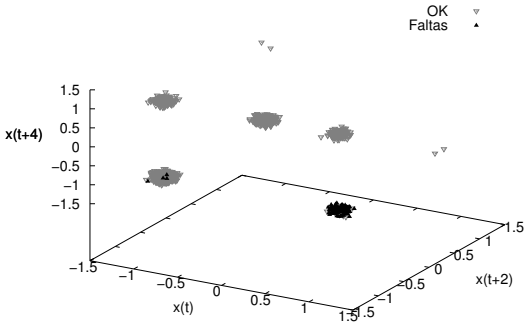


(b) Reconstrução do *trace* 22

Figura A.22: *Trace* 22 do utilitário rsync, juntamente com sua reconstrução.

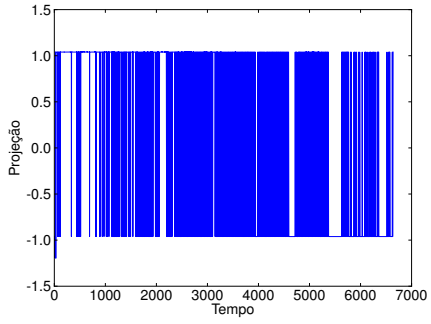


(a) Projeção do *trace* 23.

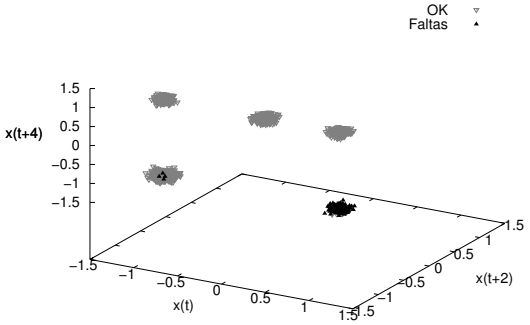


(b) Reconstrução do *trace* 23

Figura A.23: *Trace* 23 do utilitário rsync, juntamente com sua reconstrução.

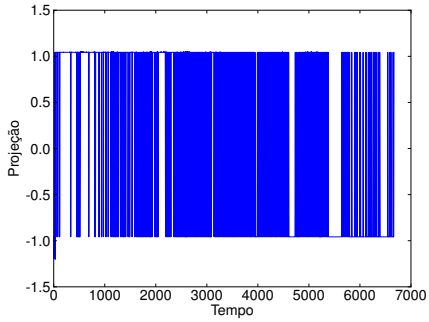


(a) Projeção do *trace* 24.

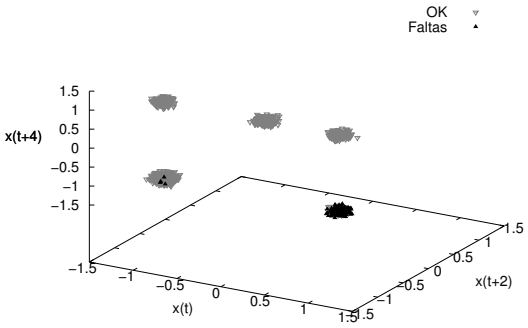


(b) Reconstrução do *trace* 24

Figura A.24: *Trace* 24 do utilitário rsync, juntamente com sua reconstrução.

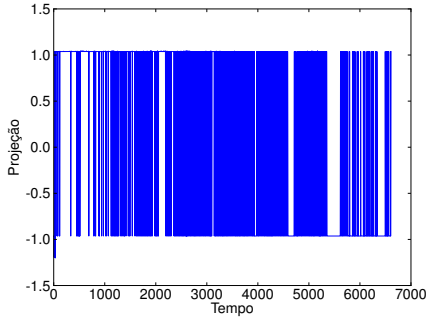


(a) Projeção do *trace* 25.

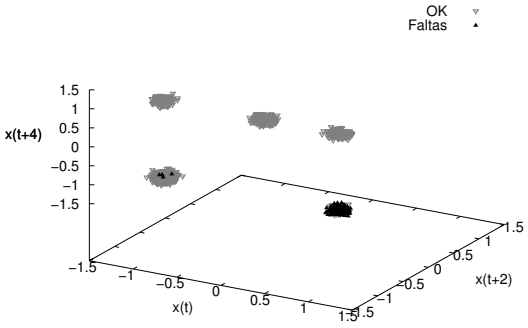


(b) Reconstrução do *trace* 25

Figura A.25: *Trace* 25 do utilitário rsync, juntamente com sua reconstrução.

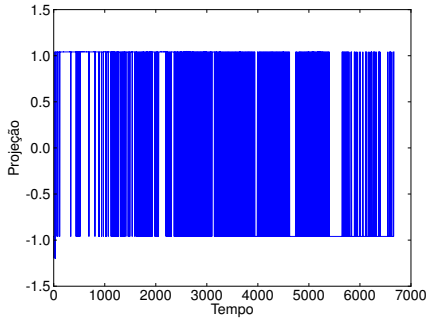


(a) Projeção do *trace* 26.

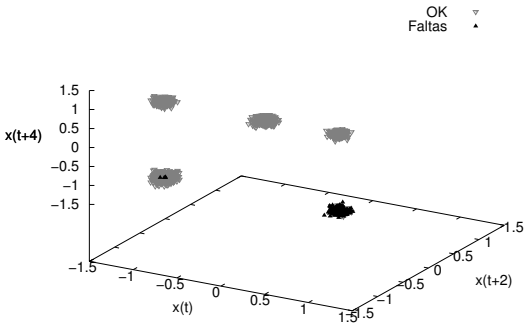


(b) Reconstrução do *trace* 26

Figura A.26: *Trace* 26 do utilitário rsync, juntamente com sua reconstrução.

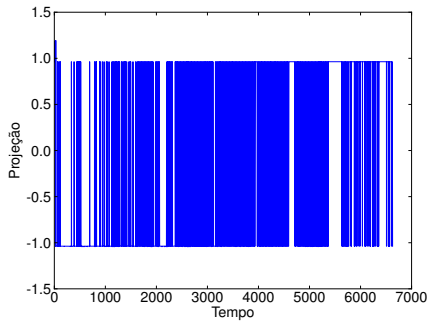


(a) Projeção do *trace* 27.

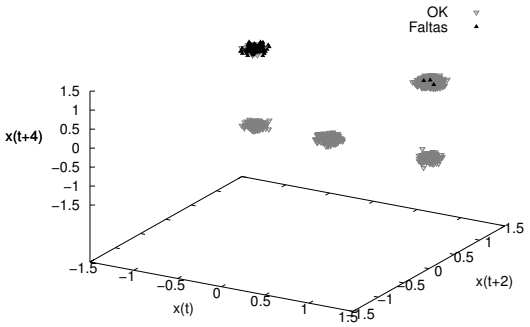


(b) Reconstrução do *trace* 27

Figura A.27: *Trace* 27 do utilitário rsync, juntamente com sua reconstrução.

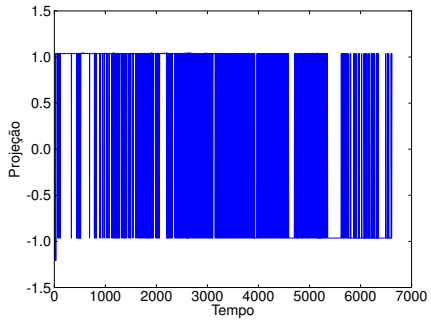


(a) Projeção do *trace* 28.

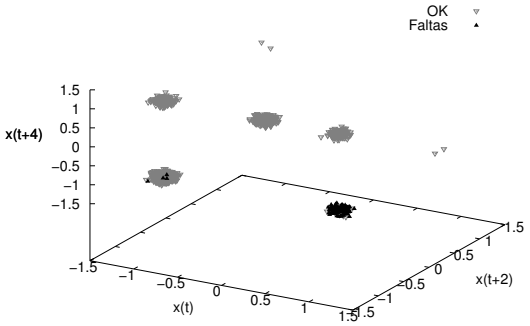


(b) Reconstrução do *trace* 28

Figura A.28: *Trace* 28 do utilitário rsync, juntamente com sua reconstrução.

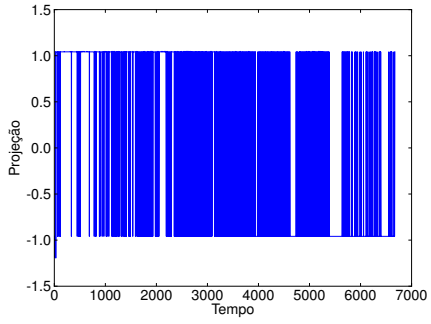


(a) Projeção do *trace* 29.

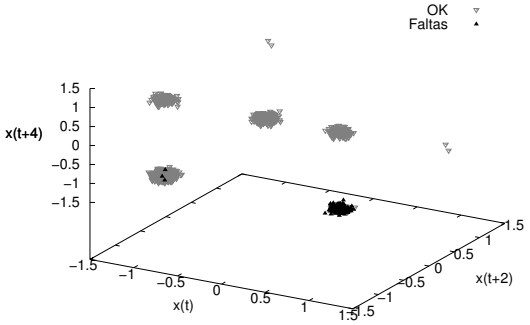


(b) Reconstrução do *trace* 29

Figura A.29: *Trace* 29 do utilitário rsync, juntamente com sua reconstrução.



(a) Projeção do *trace* 30.



(b) Reconstrução do *trace* 30

Figura A.30: *Trace* 30 do utilitário rsync, juntamente com sua reconstrução.

Referências

ABARBANEL, H.; BROWN, R.; SIDOROWICH, J.; TSIMRING, L. The analysis of observed chaotic data in physical systems. *Reviews of Modern Physics*, v. 65, n. 4, p. 1331–1392, 1993. Citado na página 69.

ALBERTINI, M. K. *Medição de novidades em séries temporais utilizando aprendizado de máquina incremental*. Relatório Técnico, Universidade de São Paulo, 2009. Citado na página 45.

ALBERTINI, M. K.; DE MELLO, R. F. *A self-organizing neural network to approach novelty detection*, cap. 1. IGI Global, p. 1–21, 2008. Citado na página 54.

ALLIGOOD, K.; SAUER, T.; YORKE, J. *Chaos: an introduction to dynamical systems*. Springer, 1997. Citado nas páginas 67 e 68.

ARORA, A.; GOUDA, M. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, v. 19, n. 11, p. 1015–1027, 1993. Citado na página 7.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, v. 1, n. 1, p. 11–33, 2004. Citado nas páginas 3, 7, e 8.

BALAZINSKA, M.; BALAKRISHNAN, H.; MADDEN, S. R.; STONEBRAKER, M. Fault-tolerance in the borealis distributed stream processing system. *ACM Transactions on Database Systems*, v. 33, n. 1, p. 1–44, 2008. Citado na página 10.

BISHOP, C. M. *Pattern recognition and machine learning*. 1st ed. Springer, 2007. Citado na página 35.

BONAKDARPOUR, B.; KULKARNI, S. S. Masking faults while providing bounded-time phased recovery. In: *FM '08: Proceedings of the 15th in-*

ternational symposium on Formal Methods, Berlin, Heidelberg: Springer, p. 374–389, 2008. Citado na página 10.

BURGES, C. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, v. 2, n. 2, p. 121–167, 1998. Citado nas páginas 24 e 25.

CAMINERO, A.; SULISTIO, A.; CAMINERO, B.; CARRION, C.; BUYYA, R. Extending gridsim with an architecture for failure detection. *International Conference on Parallel and Distributed Systems*, v. 1, p. 1–8, 2007. Citado nas páginas 13 e 14.

CHANDRA, T. D.; HADZILACOS, V.; TOUEG, S. The weakest failure detector for solving consensus. *Journal of the ACM*, v. 43, n. 4, p. 685–722, 1996. Citado na página 3.

CHEATHAM, T.; FAHMY, A.; STEFANESCU, D.; VALIANT, L. Bulk synchronous parallel computing-a paradigm for transportable software. *Hawaii International Conference on System Sciences*, v. 0, p. 268, 1995. Citado na página 2.

CHING, W.-K.; ZHANG, S.; NG, M. K.; AKUTSU, T. An approximation method for solving the steady-state probability distribution of probabilistic Boolean networks. *Bioinformatics*, v. 23, n. 12, p. 1511–1518, 2007. Citado na página 37.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Introduction to algorithms*. 2nd ed. Cambridge, Massachusetts: The MIT Press, 2001. Citado na página 46.

COVER, T. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, v. 14, n. 3, p. 326–334, 1965. Citado na página 32.

DAVIS, P. *Interpolation and approximation*. Blaisdell, 1963. Citado na página 33.

DEGA, J. L. The redundancy mechanisms of the ariane 5 operational control center. In: *FTCS '96: Proceedings of the The Twenty-Sixth Annual International Symposium on Fault-Tolerant Computing (FTCS '96)*, Washington, DC, EUA: IEEE Computer Society, p. 382, 1996. Citado na página 10.

DEVARAKONDA, M. V.; IYER, R. K. Predictability of process resource usage: A measurement-based study on unix. *IEEE Transactions on Software Engineering*, v. 15, n. 12, p. 1579–1586, 1989. Citado na página 5.

DIJKSTRA, E. W. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, v. 17, n. 11, p. 643–644, 1974. Citado na página 10.

DODONOV, E.; DE MELLO, R. F. A model for automatic on-line process behavior extraction, classification and prediction in heterogeneous distributed systems. In: *CCGRID*, p. 899–904, 2007. Citado na página 5.

DOUGLIS, F.; DAVISON, B. D., eds. *Web content caching and distribution: Proceedings of the 8th international workshop*. Springer, 2009. Citado na página 72.

EBNENASIR, A.; KULKARNI, S. S.; ARORA, A. Ftsyn: a framework for automatic synthesis of fault-tolerance. *International Journal on Software Tools for Technology Transfer*, v. 10, n. 5, p. 455–471, 2008. Citado na página 10.

EL-SHISHINY, H.; DERAZ, S.; BAHY, O. Mining software aging patterns by artificial neural networks. In: *ANNPR '08: Proceedings of the 3rd IAPR workshop on Artificial Neural Networks in Pattern Recognition*, Berlin, Heidelberg: Springer, p. 252–262, 2008. Citado nas páginas 4 e 21.

ERNST, M. D.; PERKINS, J. H.; GUO, P. J.; MCCAMANT, S.; PACHECO, C.; TSCHANTZ, M. S.; XIAO, C. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, v. 69, n. 1-3, p. 35–45, 2007. Citado na página 22.

FALAI, L.; BONDAVALLI, A. Experimental evaluation of the qos of failure detectors on wide area network. In: *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks*, Washington, DC, EUA: IEEE Computer Society, p. 624–633, 2005. Citado na página 20.

FENG, T. H.; LEE, E. A. Real-time distributed discrete-event execution with fault tolerance. In: *RTAS '08: Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, Washington, DC, EUA: IEEE Computer Society, p. 205–214, 2008. Citado na página 1.

FILHO, F. C.; MARQUES, A.; DE CAMARGO, R. Y.; KON, F. A group membership service for large-scale grids. In: *MGC '08: Proceedings of the 6th international workshop on Middleware for grid computing*, Nova Iorque, NY, EUA: ACM, p. 1–6, 2008. Citado nas páginas 1, 3, 11, e 14.

FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. S. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, v. 32, n. 2, p. 374–382, 1985. Citado na página 3.

FISHER, R. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, v. 7, p. 179–188, 1936. Citado na página 52.

FOSTER, I.; ET AL. The grid: A new infrastructure for 21st century science. *Physics Today*, v. 55, n. 2, p. 42–47, 2002. Citado na página 1.

FRASER, A.; SWINNEY, H. Independent coordinates for strange attractors from mutual information. *Physical Review A*, v. 33, n. 2, p. 1134–1140, 1986. Citado nas páginas 70 e 74.

FREEMAN, J. A.; SKAPURA, D. M. *Neural networks: algorithms, applications, and programming techniques*. Redwood City, CA, EUA: Addison Wesley Longman Publishing Co., Inc., 1991. Citado na página 37.

GÄRTNER, F. C. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, v. 31, n. 1, p. 1–26, 1999. Citado nas páginas xvii, 2, 3, 7, 8, 9, e 10.

HANSON, S. J.; KRAUT, R. E.; FARBER, J. M. Interface design and multivariate analysis of unix command use. *ACM Transactions on Information Systems*, v. 2, n. 1, p. 42–57, 1984. Citado na página 55.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The elements of statistical learning: Data mining, inference, and prediction*. 2a ed. Springer, 2010. Citado na página 20.

HAYASHIBARA, N.; CHERIF, A.; KATAYAMA, T. Failure detectors for large-scale distributed systems. In: *SRDS '02: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, Washington, DC, EUA: IEEE Computer Society, p. 404, 2002. Citado nas páginas 3, 11, 12, e 14.

HAYASHIBARA, N.; DEFAGO, X.; YARED, R.; KATAYAMA, T. The phi accrual failure detector. In: *SRDS '04: Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems*, Washington, DC, EUA: IEEE Computer Society, p. 66–78, 2004. Citado na página 14.

HAYKIN, S. *Neural networks: a comprehensive foundation*. Upper Saddle River, NJ, EUA: Prentice Hall PTR, 2008. Citado nas páginas xiv, 31, 32, 33, 34, e 41.

HEGGER, R.; KANTZ, H.; SCHREIBER, T. Practical implementation of nonlinear time series methods: The tisean package. *CHAOS*, v. 9, p. 413, 1999. Citado na página 74.

HOFFMANN, G.; TRIVEDI, K.; MALEK, M. A best practice guide to resource forecasting for computing systems. *IEEE Transactions on Reliability*, v. 56, n. 4, p. 615–628, 2007. Citado nas páginas 4, 22, 63, e 66.

HOFFMANN, G. A.; TRIVEDI, K. S.; MALEK, M. A best practice guide to resources forecasting for the apache webserver. In: *PRDC '06: Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, Washington, DC, EUA: IEEE Computer Society, p. 183–193, 2006. Citado na página 5.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, v. 2, n. 5, p. 359–366, 1989. Citado na página 22.

HSUEH, M.-C.; TSAI, T. K.; IYER, R. K. Fault injection techniques and tools. *Computer*, v. 30, n. 4, p. 75–82, 1997. Citado na página 55.

HUANG, Y.; KINTALA, C.; KOLETTIS, N.; FULTON, N. D. Software rejuvenation: Analysis, module and applications. *International Symposium on Fault-Tolerant Computing*, v. 0, p. 0381, 1995. Citado nas páginas 4 e 20.

KANAWATI, G. A.; KANAWATI, N. A.; ABRAHAM, J. A. Ferrari: A flexible software-based fault and error injection system. *IEEE Transactions on Computers*, v. 44, n. 2, p. 248–260, 1995. Citado nas páginas 54 e 55.

KAO, W.-L.; IYER, R. K.; TANG, D. Fine: A fault injection and monitoring environment for tracing the unix system behavior under faults. *IEEE Transactions on Software Engineering*, v. 19, n. 11, p. 1105–1118, 1993. Citado na página 5.

KENNEL, M.; BROWN, R.; ABARBANEL, H. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical Review A*, v. 45, n. 6, p. 3403–3411, 1992. Citado nas páginas 70 e 71.

KEOGH, E.; RATANAMAHATANA, C. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, v. 7, n. 3, p. 358–386, 2005. Citado nas páginas 58 e 60.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, v. 36, n. 1, p. 41–50, 2003. Citado na página 20.

KON, F.; GOLDMAN, A. *Grades computacionais: Conceitos fundamentais e casos concretos*, cap. 2 Belém do Pará: Sociedade Brasileira de Computação, p. 55–104, 2008. Citado na página 1.

LAMPORT, L.; SHOSTAK, R.; PEASE, M. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, v. 4, n. 3, p. 382–401, 1982. Citado na página 3.

LI, L.; VAIDYANATHAN, K.; TRIVEDI, K. S. An approach for estimation of software aging in a web server. In: *ISESE '02: Proceedings of the 2002 International Symposium on Empirical Software Engineering*, Washington, DC, EUA: IEEE Computer Society, p. 91, 2002. Citado nas páginas 4, 19, 20, e 63.

LONG, D.; MUIR, A.; GOLDING, R. A longitudinal survey of internet host reliability. In: *SRDS '95: Proceedings of the 14TH Symposium on Reliable Distributed Systems*, Washington, DC, EUA: IEEE Computer Society, p. 2, 1995. Citado na página 13.

LORENA, A. C.; CARVALHO, A. C. P. L. F. Uma introdução às support vector machines. *Revista de Informática Teórica e Aplicada*, v. XIX, p. 43–67, 2007. Citado nas páginas xiv, 22, 24, 25, 26, 27, 28, 29, e 63.

LORENZ, E. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 1963. Citado na página 69.

LU, S.; HALANG, W. Incorporating Fault Tolerance into Component-based Architectures for Embedded Systems. *Journal of Automation, Mobile Robotics & Intelligent Systems*, v. 3, n. 1, 2009. Citado na página 10.

MEDEIROS, R.; CIRNE, W.; BRASILEIRO, F.; SAUVÉ, J. Faults in grids: Why are they so bad and what can be done about it? In: *GRID '03: Proceedings of the 4th International Workshop on Grid Computing*, Washington, DC, EUA: IEEE Computer Society, p. 18, 2003. Citado na página 2.

DE MELLO, R.; DODONOV, E.; BERTAGNA, R.; SENGER, L. Extracting and predicting the communication behaviour of parallel applications. *International Journal of Parallel, Emergent and Distributed Systems*, v. 24, n. 3, p. 225–242, 2009. Citado na página 5.

DE MELLO, R. F. Sistemas dinâmicos e técnicas inteligentes para a predição de comportamento de processos: uma abordagem para otimização de escalonamento em grades computacionais. Tese de livre docência, 2009. Citado nas páginas 70 e 71.

MHASKAR, H. N. Neural networks for optimal approximation of smooth and analytic functions. *Neural Comput.*, v. 8, n. 1, p. 164–177, 1996. Citado na página 32.

MICCHELLI, C. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, v. 2, n. 1, p. 11–22, 1986. Citado nas páginas 34 e 35.

MITCHELL, T. *Machine learning*. 1st ed. McGraw Hill Higher Education, 1997. Citado na página 17.

MORETTIN, P. A.; TOLOI, C. M. C. *Análise de séries temporais*. 2a edição ed. Associação Brasileira de Estatística (ABE), 2006. Citado nas páginas xiii, 15, 16, 17, e 19.

MULLER, K.; MIKA, S.; RATSCH, G.; TSUDA, K.; SCHOLKOPF, B. An introduction to kernel-based learning algorithms. *IEEE transactions on neural networks*, v. 12, n. 2, p. 181–201, 2001. Citado nas páginas 23 e 24.

NING, M. H.; YONG, Q.; DI, H.; YING, C.; ZHONG, Z. J. Software aging prediction model based on fuzzy wavelet network with adaptive genetic algorithm. In: *ICTAI '06: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, Washington, DC, EUA: IEEE Computer Society, p. 659–666, 2006. Citado nas páginas 4 e 21.

NURMI, D.; BREVIK, J.; WOLSKI, R. Modeling machine availability in enterprise and wide-area distributed computing environments. In: *Euro-Par 2005 Parallel Processing*, Springer, p. 432–441, 2005 (*Lecture Notes in Computer Science*, v.3648/2005). Citado nas páginas 11, 13, 14, e 72.

PADALA, P. Playing with ptrace, part ii. *Linux Journal*, v. 2002, n. 104, p. 4, 2002. Citado na página 55.

PASSERINI, A. *Kernel methods, multiclass classification and applications to computational molecular biology*. Tese de Doutoramento, Università degli Studi di Trento, 2004. Citado nas páginas 23 e 24.

PERKINS, J. H.; KIM, S.; LARSEN, S.; AMARASINGHE, S.; BACHRACH, J.; CARBIN, M.; PACHECO, C.; SHERWOOD, F.; SIDIROGLOU, S.; SULLIVAN, G.; WONG, W.-F.; ZIBIN, Y.; ERNST, M. D.; RINARD, M. Automatically patching errors in deployed software. In: *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, Nova Iorque, NY, EUA: ACM, p. 87–102, 2009. Citado na página 22.

POWELL, J. D. Radial basis function approximations to polynomials, p. 223–241. 1988. Citado na página 33.

REN, X.; LEE, S.; EIGENMANN, R.; BAGCHI, S. Resource availability prediction in fine-grained cycle sharing systems. In: *In Proceedings of the 15th*

IEEE International Symposium on High Performance Distributed Computing, p. 93–104, 2006. Citado na página 21.

VAN RENESSE, R.; MINSKY, Y.; HAYDEN, M. A gossip-style failure detection service. In: *Middleware*, p. 55–70, 1998. Citado nas páginas 3 e 12.

ROSS, S. M. *Stochastic processes*. 2 ed. Academic Press, 1995. Citado na página 41.

ROSS, S. M. *Introduction to probability models*. 8 ed. Academic Press, 2003. Citado nas páginas 35 e 36.

SAHOO, R. K.; OLINER, A. J.; RISH, I.; GUPTA, M.; MOREIRA, J. E.; MA, S.; VILALTA, R.; SIVASUBRAMANIAM, A. Critical event prediction for proactive management in large-scale computer clusters. In: *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, Nova Iorque, NY, EUA: ACM, p. 426–435, 2003. Citado na página 20.

SALFNER, F.; HOFFMANN, G.; MALEK, M. Prediction-based software availability enhancement. *Self-star Properties in Complex Information Systems*, p. 143–157, 2005. Citado nas páginas 80, 81, e 82.

SAYOOD, K. *Lossless compression handbook*. 1a ed. Academic Press, 2003. Citado na página 72.

SCHROEDER, B.; GIBSON, G. A. A large-scale study of failures in high-performance computing systems. *International Conference on Dependable Systems and Networks*, v. 0, p. 249–258, 2006. Citado na página 1.

SHANNON, C. E. A mathematical theory of communication. *SIGMOBILE Mobile Computing and Communications Review*, v. 5, n. 1, p. 3–55, 2001. Citado nas páginas 38 e 41.

SMOLA, A. *Advances in large margin classifiers*. the MIT Press, 2000. Citado nas páginas 26, 27, 28, e 29.

SRIDHARAN, M.; BAPAT, S.; RAMNATH, R.; ARORA, A. Implementing an autonomic architecture for fault-tolerance in a wireless sensor network testbed for at-scale experimentation. In: *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, Nova Iorque, NY, EUA: ACM, p. 1670–1676, 2008. Citado na página 10.

TAKENS, F. Detecting strange attractors in turbulence. *Dynamical systems and turbulence, Warwick 1980*, p. 366–381, 1981. Citado na página 69.

TANENBAUM, A. S.; STEEN, M. V. *Sistemas distribuídos princípios e paradigmas*. São Paulo: Pearson, 2007. Citado na página 1.

TORMENE, P.; GIORGINO, T.; QUAGLINI, S.; STEFANELLI, M. Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation. *Artificial Intelligence in Medicine*, v. 45, n. 1, p. 11–34, 2009. Citado na página 60.

TURAU, V.; WEYER, C. Fault tolerance in wireless sensor networks through self-stabilisation. *International Journal of Communication Networks and Distributed Systems*, v. 2, n. 1, p. 78–98, 2009. Citado na página 10.

TURNBULL, D.; ALLDRIN, N. Failure Prediction in Hardware Systems. *UCSD CSE221 Project*, 2003. Citado nas páginas 4 e 21.

VAPNIK, V. *The nature of statistical learning theory*. Springer, 2000. Citado nas páginas 23 e 25.

VAPNIK, V.; CHERVONENKIS, A. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, v. 16, p. 264, 1971. Citado na página 24.

WALPOLE, R. E.; MYERS, R. H.; MYERS, S. L.; YE, K. *Probability & statistics for engineers & scientists*. 8a ed. Prentice Hall, 2006. Citado nas páginas 17, 51, e 63.

WEEDON, G. *Time-series analysis and cyclostratigraphy: examining stratigraphic records of environmental cycles*. Cambridge University Press, 2003. Citado na página 16.

WHITNEY, H. Differentiable manifolds. *Annals of Mathematics*, v. 37, n. 3, p. 645–680, 1936. Citado na página 68.

WITTEN, I. H.; FRANK, E. *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann Series in Data Management Systems, second ed. Morgan Kaufmann, 2005. Citado nas páginas 49 e 73.

XUE, Z.; DONG, X.; MA, S.; DONG, W. A survey on failure prediction of large-scale server clusters. In: *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, p. 733–738, 2007. Citado nas páginas 4, 5, 15, 63, e 66.

ZOU, B.; LIU, Q. ARMA-based traffic prediction and overload detection of network. *Journal of Computer Research and Development*, v. 39, n. 12, p. 1645–1652, 2002. Citado nas páginas 20 e 66.