5
Robustez e Diagnóstico de Problemas de Desempenho

Neste capítulo, são abordadas as outras duas perspectivas que completam a caracterização de problemas de desempenho. Primeiramente, a robustez do mecanismo de previsão de problemas desenvolvido no capítulo anterior é analisada e uma forma de aumentar o poder de estimação do classificador e reduzir a emissão de alarmes falsos é proposta. O objetivo, neste caso, é responder a questão $\mathbf{Q_2}$ formulada no Capítulo 1 e obter um mecanismo de previsão mais robusto a falhas transientes do classificador. Segundo, são investigadas formas de classificar o nível de influência de cada métrica observada em um problema de desempenho particular. O objetivo é responder a questão $\mathbf{Q_3}$, também definida no Capítulo 1, e obter um mecanismo de diagnóstico independente do algoritmo ou do classificador utilizado no mecanismo de previsão. Em ambos os casos, é proposto o uso de *testes estatísticos*.

5.1
Alarmes Falsos em Métodos de Classificação

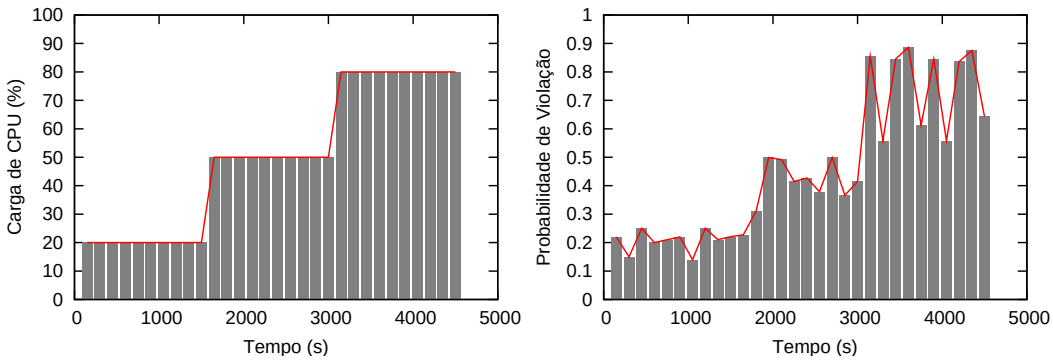
Os experimentos realizados no Capítulo 4 deste trabalho evidenciam que a taxa de alarmes falsos emitidos por métodos de classificação não é negligenciável. Esse fato é justificado pelo procedimento empregado na construção dos classificadores, o qual se baseia em um *dataset* de treinamento.

Particularmente, no contexto de caracterização de problemas de desempenho de sistemas computacionais, existem duas abordagens para a construção desses *datasets*: *benchmarks* sintéticos padrões ou *datasets* extraídos diretamente de um ambiente real ou de produção. Em geral, *benchmarks* sintéticos não são suficientemente ricos para reproduzir a amplitude de condições que podem acontecer na prática (Cohen et al., 2004) e, portanto, *datasets* de treinamento gerados nessas condições exibem variações estatísticas em relação às condições reais. Por outro lado, *datasets* gerados de ambientes reais são, em geral, mais ricos que *benchmarks* sintéticos, mas estão sujeitos a ruídos que podem se manifestar de forma aleatória. Portanto, classificadores gerados a partir de *datasets* sintéticos ou reais se tornam sujeitos a falhas transientes quando introduzidos em ambientes de produção. Essas falhas podem produzir

um nível inaceitável de alarmes falsos e afetar a confiança dos administradores de sistemas na solução automatizada de estimação de problemas.

Como ilustração, é apresentado o comportamento do classificador TAN durante a execução de uma aplicação MapReduce. Nesse exemplo, o sistema é submetido a uma carga crescente de CPU e a probabilidade de violação ($P(s^{-}|\mathbf{m})$, emitida pelo classificador, é plotada em função do tempo. O gráfico da esquerda da Figura 5.1 ilustra o comportamento da carga durante 75 minutos de execução. Cada ponto no gráfico representa a média da carga de 10 amostras, sendo as amostras coletadas a cada 15 segundos. O gráfico da direita da Figura 5.1 representa a probabilidade de violação emitida pelo classificador durante os 75 minutos de execução. Cada ponto nesse gráfico é a média de 10 probabilidades, cada uma das quais emitidas aproximadamente a cada 15 segundos.

Figura 5.1: Comportamento da probabilidade de violação emitida por um classificador em um sistema submetido a uma carga crescente.



Duas observações podem ser feitas a partir da Figura 5.1. Primeiro, é possível notar que a probabilidade de violação aumenta à medida que a intensidade da carga aumenta. Logo, um padrão crescente de probabilidade de violação é um indicativo de problema de desempenho. Segundo, apesar da curva que representa a probabilidade de violação mostrar uma tendência de crescimento, a curva não é monotônica, uma vez que o classificador está sujeito a falhas.

Essas observações motivaram a investigação do uso de testes estatísticos para monitorar as saídas produzidas pelo classificador. O intuito é usar testes estatísticos que detectam tendência em séries temporais para estimar um problema de desempenho. Dessa forma, não apenas o valor imediato da última previsão do classificador é considerado, mas também a tendência de crescimento da curva formada pelas estimativas mais recentes. Como um

padrão crescente de probabilidade de violação é um indicativo de problemas futuros de desempenho, técnicas de detecção de tendência podem capturar esse padrão, mesmo diante de eventuais alarmes falsos, como será mostrado a seguir.

5.1.1 Algoritmo Robusto de Alerta de Problema de Desempenho

Para monitorar a tendência de crescimento da probabilidade de violação e atenuar os efeitos de alarmes falsos no comportamento do classificador, neste trabalho, é usado um teste denominado *arranjo reverso* (Bendat e Piersol, 1986). Esse teste é empregado com o intuito de detectar, de forma rápida e simples, tendências em séries temporais.

Considere p_i, $i = 1..N$, uma sequência de observações da variável P. Cada p_i representa uma probabilidade de violação emitida pelo classificador no tempo t_i. De fato, p_i representa uma sequência formada pelos valores mais recentes estimados para $P(s^-|\mathbf{m})$. Dada tal sequência, o teste proposto calcula a estatística $A = \sum_{i=1}^{N-1} A_i$. Esta consiste na soma de todos os arranjos reversos, onde um arranjo reverso é definido como uma ocorrência $p_i > p_j$ quando $i < j$. Para encontrar A, são usadas as somas intermediárias A_i e uma função indicativa h_{ij}, como mostra a Equação 5-1:

$$\mathrm{A} = \sum_{i=1}^{N-1} A_i, \quad \text{onde} \tag{5-1}$$

$$A_i = \sum_{j=i+1}^{N} h_{ij} \quad \text{e} \tag{5-2}$$

$$h_{ij} = \begin{cases} 1 & \text{se } p_i > p_j \\ 0 & \text{caso contrário.} \end{cases} \tag{5-3}$$

Como exemplo de como computar a estatística A, considere a sequência $p_i = \{0.1, 0.4, 0.3, 0.7, 0.2, 0.8, 0.6, 1.0, 0.9, 0.5\}$. Os valores de A_i, para $i = 1, 2, .., 9$, são calculados abaixo:

$$A_1 = \sum_{j=2}^{10} h_{1j} = 0, \quad A_2 = \sum_{j=3}^{10} h_{2j} = 2, \quad ... \quad A_9 = \sum_{j=9}^{10} h_{9j} = 1.$$

Após o calculo das somas intermediárias, obtém-se o vetor $[A_i] = [0, 2, 1, 3, 0, 2, 1, 2, 1]$. A estatística A é então calculada como a soma desses valores, ou seja, $A = 12$.

Tendo computado o valor da estatística A e considerando um nível de confiança α, o teste de arranjo reverso estabelece que a sequência p_i é livre de tendência se A encontra-se no intervalo:

$$A_{N;1-\alpha/2} \leq A \leq A_{N;\alpha/2}. \tag{5-4}$$

A Equação 5-4 indica o intervalo de valores no qual o número de inversões ou arranjos reversos podem ser considerados como ocorridos ao acaso. Dito de outra forma, se o valor de A se encontra dentro do intervalo, a sequência analisada não apresenta tendência. Se o valor de A for maior que o limite superior do intervalo, então foram encontradas mais inversões que o esperado para uma situação regida ao acaso e considera-se que a sequência analisada apresenta uma tendência de decrescimento. Analogamente, quando o valor de A é menor que o limite inferior do intervalo, então foram encontradas menos inversões que o esperado para uma situação regida ao acaso e considera-se que a sequência analisada apresenta uma tendência de crescimento.

Além do teste de arranjo reverso descrito acima, um método de suavização também é aplicado para atenuar os efeitos produzidos por variações aleatórias na sequência p_i. Nesse caso, é proposta uma *média móvel ponderada* ou WMA. Dada a sequência p_i, calcula-se a WMA como:

$$WMA = \frac{\sum_{i=1}^{N} i * p_i}{\sum_{i=1}^{N} i}. \tag{5-5}$$

O algoritmo 5.1 representa a proposta deste trabalho para um mecanismo de alerta de problemas de desempenho mais robusto a alarmes falsos. Nesse mecanismo, um classificador tradicional é combinado com o teste de arranjo reverso e o método de suavização da seguinte forma. Uma sequência das últimas N probabilidades de violação emitidas pelo classificador é mantida em memória. Sempre que uma nova probabilidade de violação é emitida pelo classificador, ela substitui a probabilidade mais antiga, mantida na memória, e o novo conjunto é usado como entrada para o teste de arranjo reverso. O novo conjunto é usado também para computar a WMA. O algoritmo emite um sinal de alerta de problema de desempenho em duas situações: *i*) se WMA atinge um limiar particular ($thUpper$) ou; *ii*) se o teste de arranjo reverso detecta uma tendência de crescimento da probabilidade de violação a partir de um certo limiar ($thRag$).

Similarmente, o algoritmo interrompe o sinal de alerta em duas situações: *i*) quando WMA atinge um limiar particular ($thLower$) ou; *ii*) quando o teste de arranjo reverso detecta uma tendência de decrescimento da probabilidade de violação a partir de um certo limiar ($thRag$). É importante notar que o nível de

confiança α do teste de arranjo reverso e os limiares $thUpper$, $thLower$ e $thRag$ são os parâmetros que controlam o compromisso entre robustez e detecção rápida de degradação de desempenho. O ajuste desses parâmetros pode levar o algoritmo a trabalhar de forma mais ou menos conservadora em relação à percepção de um problema de desempenho. A seguir, são apresentados alguns experimentos em que o algoritmo de alarme de problemas de desempenho é avaliado.

Algoritmo 5.1 Algoritmo de alerta de problemas de desempenho robusto a alarmes falsos.

Require: Uma sequencia temporal p_i dos N valores mais recentes estimados para a probabilidade de violação. Limiares $thUpper$, $thLower$ and $thRag$. Nível de significância α.

Ensure: alarm:0 (desligado) ou alarm:1 (ligado)

1: Given p_i, $i = 1...N$, compute A using Equation (5-1)
2: Given p_i, $i = 1...N$, compute WMA using Equation (5-3)
3: **if** $WMA \geq thUpper$ or ($p_N \geq thRag$ and $A < A_{N;1-\alpha/2}$) **then**
4: alarm = 1;
5: **else if** $WMA \leq thLower$ or ($p_N \leq thRag$ and $A > A_{N;\alpha/2}$) **then**
6: alarm = 0;
7: **end if**

5.1.2 Experimentos

Nesta seção, é apresentada uma avaliação do algoritmo de alerta de desempenho proposto neste trabalho. Inicialmente, o intuito é verificar como o algoritmo se comporta em dois cenários de cargas diferentes. Em seguida, o comportamento do algoritmo de alerta é avaliado com o *dataset* MapReduce. O intuito desse último experimento é comparar o desempenho do algoritmo de alerta e o desempenho apresentado pelos classificadores investigados no Capítulo 4. Por fim, o desempenho do SMART é avaliado quando o algoritmo de alerta é implementado como parte de sua máquina de inferência.

Carga Senoidal e Escada

Para avaliar a efetividade do algoritmo de alerta, este trabalho procurou por padrões de carga que fossem, ao mesmo tempo, conhecidos na literatura e que pudessem avaliar o mecanismo em diferentes condições. Foram adotados, então, dois padrões sugeridos por Cohen et al. em (Cohen et al., 2004).

Primeiramente, um padrão de carga de CPU cujo comportamento se assemelha a uma senóide foi aplicado ao sistema executando a aplicação *WordCount*. O tempo de duração do experimento foi de aproximadamente

2 horas de execução, tendo a carga de CPU, durante esse período, oscilado para níveis baixos e altos repetidamente. A intensão desse padrão é simular rajadas pequenas e repentinas de carga. A Tabela 5.1 resume as configurações usadas no experimento. O SLO é definido para a variável *cpu_usage* e o valor do limiar é configurado em 50% de utilização. O nível de significância para o teste arranjo reverso é configurado em 0.01. $thUpper$, $thLower$, $thRag$ são configurados em 0.5, 0.2 e 0.4, respectivamente. Essa configuração reflete um comportamento moderado para o algoritmo, isto é, garante maior robustez ao algoritmo sem torná-lo muito reativo. A classificação do sistema é feita levando em consideração as 10 probabilidades de violação mais recentes emitidas pelo classificador, ou seja, N é configurado com o valor 10.

Tabela 5.1: Configuração dos parâmetros usados no algoritmo de alerta de problemas de desempenho quando o sistema é submetido a um padrão de carga senoidal.

Parâmetro	**Configuração**
Classificador	TAN
Violação de SLO	ocorrência da variável *cpu_usage* que excede 50% de utilização
α	0.01
$thUpper$	0.5
$thLower$	0.2
$thRag$	0.4
N	10

A Figura 5.2 mostra: a carga aplicada ao sistema, os alarmes falsos emitidos pelo classificador, a saída final do algoritmo de alerta e o valor limiar para o SLO, durante as 2 horas de execução. Todas essas informações são mostradas em função do tempo. É possível notar que enquanto a carga oscila rapidamente, os valores imediatos emitidos pelo classificador também oscilam e alarmes falsos são produzidos por este. O algoritmo de alerta, no entanto, é capaz de capturar a tendência predominante e nenhum alarme falso é produzido por ele. A Figura 5.2 ilustra o momento em que o alarme do algoritmo de alerta é ligado. Isso ocorre apenas poucos segundos depois da utilização de CPU atingir ou exceder o limiar. Por outro lado, o alarme do algoritmo de alerta é desligado apenas poucos segundos após a carga voltar a normalidade. Esse comportamento demonstra a natureza mais conservadora do algoritmo, cuja origem está relacionada à amortização causada pela WMA.

Usando a mesma configuração definida na Tabela 5.1, o segundo padrão de carga de CPU é aplicado ao sistema executando a aplicação *WordCount*. O comportamento desse padrão se assemelha a uma escada, como mostra a

Figura 5.2: Comportamento do algoritmo de alerta de problemas de desempenho quando o sistema é submetido a um padrão de carga senoidal.

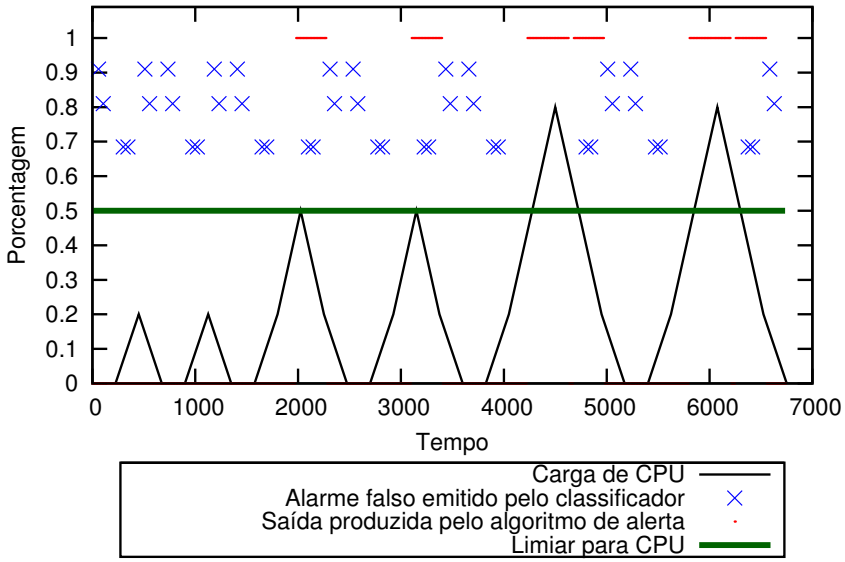
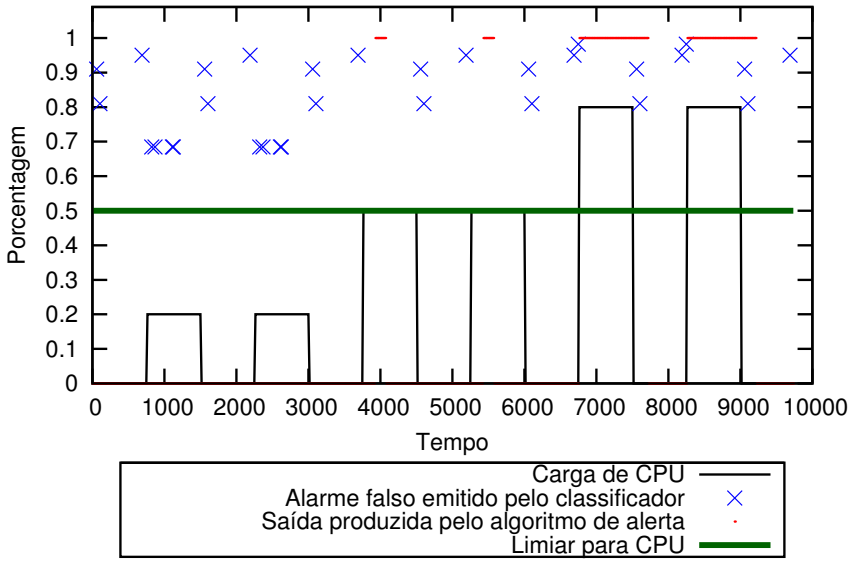


Figura 5.3. O experimento durou aproximadamente 2.5 horas, período durante o qual a carga de CPU oscilou entre rajadas de 12 minutos de atividade, seguidas de rajadas de igual tamanho de inatividade. A cada duas rajadas de atividade, a intensidade da carga aumentava. O intuito desse padrão é simular rajadas repentinas, porém sustentáveis, de carga crescente de CPU. A Figura 5.3 ilustra o resultado desse experimento. Novamente, são plotados: a carga aplicada ao sistema, os alarmes falsos emitidos pelo classificador, a saída final do algoritmo de alerta e o valor limiar para o SLO, durante o tempo de execução do experimento. É possível notar que apesar do algoritmo de alarme não emitir alarmes falsos, ele não foi capaz de manter o alarme ativado durante todo o período em que a CPU atingiu 50% de utilização. A causa desse mau comportamento pode ser explicada pela intensidade da carga a qual, no intervalo em questão, era exatamente a fronteira entre os estados de violação e conformidade. Isso explica porque o classificador tem um desempenho pior nesse intervalo. No entanto, quando a intensidade de carga aumenta e se afasta da fronteira, o algoritmo de alarme funciona como esperado.

Os experimentos acima permitem analisar o comportamento do algoritmo de alarme em duas situações diferentes: quando a intensidade da carga oscila entre os níveis alto e baixo rapidamente e quando ela oscila entre os estados de atividade e ausência de atividade de forma sustentável. Em ambos os casos, nota-se que o algoritmo de alarme proposto atinge o objetivo esperado de atenuação dos efeitos dos alarmes falsos emitidos pelo classificador.

Figura 5.3: Comportamento do algoritmo de alerta de problemas de desempenho quando o sistema é submetido a um padrão de carga do tipo escada.



Dataset MapReduce

A acurácia do algoritmo de alerta para estimar um problema de desempenho também foi avaliada. Analogamente, avaliou-se o tempo requerido para o algoritmo treinar um modelo e classificar uma instância. Os resultados obtidos foram comparados com os valores obtidos pelos classificadores nos experimentos do Capítulo 4. As configurações usadas neste experimento estão resumidas na Tabela 5.2. Como nos experimentos do Capítulo 4, o SLO é definido para a variável *avg_response_time*. O valor do limiar para o SLO é definido como o 50ºpercentil da variável. O nível de significância para o teste arranjo reverso é configurado em 0.01. Diferentemente dos cenários de carga senoidal e escada, no *dataset* MapReduce, o melhor desempenho do algoritmo de alerta é observado com a seguinte configuração para os parâmetros *thUpper*, *thLower* e *thRag*: 0.6, 0.3 e 0.4, respectivamente. Essa configuração é usada neste experimento. A classificação do sistema é feita levando em consideração as 10 probabilidades de violação mais recentes emitidas pelo classificador, ou seja, N é configurado com o valor 10.

Os resultados para o algoritmo de alerta estão ilustrados na Tabela 5.3, a qual mostra: a taxa de detecção e alarme falso para o estado de violação, a acurácia balanceada e o tempo requerido para treinar o algoritmo. Esses valores são calculados para o algoritmo de alerta quando o mesmo é usado com cada classificador avaliado no Capítulo 4. Para efeito de comparação, uma tabela semelhante, Tabela 5.4, resume o desempenho das soluções que

utilizam os classificadores exclusivamente. Ambas as avaliações, isto é, a avaliação do algoritmo de alarme e a avaliação dos classificadores exclusivos, são feitas considerando os mesmos *dataset* de treinamento e *teste*. Esses *datasets* representam, respectivamente, $\frac{2}{3}$ e $\frac{1}{3}$ do *dataset MapReduce*.

É importante mencionar que as métricas de desempenho mostradas nas Tabelas 5.3 e 5.4 foram obtidas apenas com os arquivos de treinamento e teste. Ou seja, a técnica *10-fold cross validation* não foi aplicada neste experimento. Por esse motivo, os tempos de treinamento dos classificadores mostrados nessas tabelas são menores que os obtidos nos experimentos do Capítulo 4.

Tabela 5.2: Configuração dos parâmetros usados no algoritmo de alerta de problemas de desempenho para o teste de acurácia.

Parâmetro	**Configuração**
Classificador	NB,TAN,AODE,FBN,DT,SVM
Violação de SLO	ocorrência da variável *avg_response_time* que excede 50ºpercentil da variável
α	0.01
$thUpper$	0.6
$thLower$	0.3
$thRag$	0.4
N	10

Tabela 5.3: Teste de desempenho para o algoritmo de alerta com diferentes classificadores.

	Taxa de Detecção	**Alarme Falso**	**Acurácia**	**Tempo Treinamento**
NB	99%	21%	87%	0.1s
TAN	99%	6%	95%	0.9 s
AODE	99%	11%	92%	0.4s
FBN	99%	57%	76%	8.5s
DT	99%	10%	92%	1.6s
SVM	99%	69%	72%	3000s

Examinando as Tabelas 5.3 e 5.4, é possível notar que o algoritmo de alerta atingiu seu objetivo. Para cada classificador, o poder de estimação do algoritmo de aprendizado aumentou quando este foi combinado com o algoritmo de alerta de problemas de desempenho, como pode ser notado pelas colunas taxa de detecção e acurácia balanceada das duas tabelas. É possível notar também que a taxa de alarmes falsos diminui com o uso do algoritmo de alerta. O tempo de treinamento se manteve praticamente inalterado nos dois experimentos. Apesar de não ilustrado nas tabelas de resultado, o tempo médio requerido para classificar uma instância também foi avaliado. No entanto, para

Tabela 5.4: Teste de desempenho para os classificadores.

	Taxa de Detecção	**Alarme Falso**	**Acurácia**	**Tempo Treinamento**
NB	92%	21%	83%	0.1s
TAN	93%	19%	85%	0.8s
AODE	94%	21%	84%	0.1s
FBN	91%	69%	65%	8s
DT	93%	19%	85%	1.6s
SVM	91%	76%	65%	3000s

todos os algoritmos em ambos os experimentos, esse tempo foi negligenciável, sendo menor que 1 milissegundo. Esses experimentos demonstram que o algoritmo de alarme de problemas de desempenho, proposto neste trabalho, aumenta o poder de estimação dos algoritmos de aprendizado sem incorrer em um custo computacional maior.

Avaliação do Desempenho do SMART

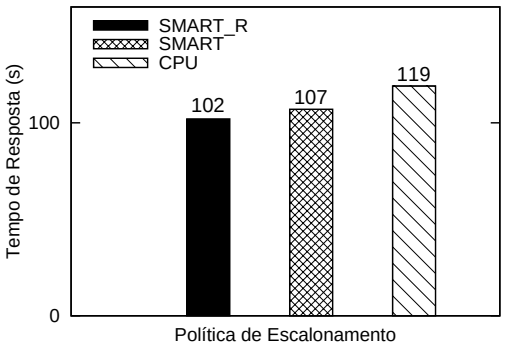
O último experimento desta seção avalia o comportamento do SMART quando sua máquina de inferência é implementada usando o algoritmo de alerta de problemas de desempenho. Conforme descrito anteriormente, a máquina de inferência do SMART estima a probabilidade de um nó violar o SLO definido. Para avaliar a efetividade do algoritmo de alerta na função de máquina de inferência do SMART, o experimento da Seção 4.6.6 foi repetido, usando os mesmos nós do *cluster*, o mesmo cenário de carga e a mesma configuração da aplicação MapReduce daquele experimento. Como na Seção 4.6.6, foi usado o classificador TAN e o SLO foi definido como o 80º percentil de *avg_response_time*.

Nas condições descritas acima, o desempenho da aplicação MapReduce foi comparado em três situações: usando a política de escalonamento baseada em CPU (CPU); usando a política baseada em probabilidade de violação, sendo a probabilidade obtida pelo uso exclusivo do classificador TAN (SMART); e usando a política baseada em probabilidade de violação, sendo a probabilidade obtida pelo algoritmo de alerta de problemas de desempenho (SMART_R). Novamente, o desempenho da aplicação é medido considerando o tempo total de execução. Para fins de comparação, considera-se a política que proporciona o melhor desempenho, aquela que resulta em um menor tempo de execução da aplicação. A Figura 5.4(a) mostra a média do tempo total de execução da aplicação MapReduce, em 10 execuções, para cada política de escalonamento, usando um nível de confiança de 95% no intervalo de confiança. A Figura 5.4(b) ilustra o valor médio do tempo de execução e o intervalo de confiança para cada

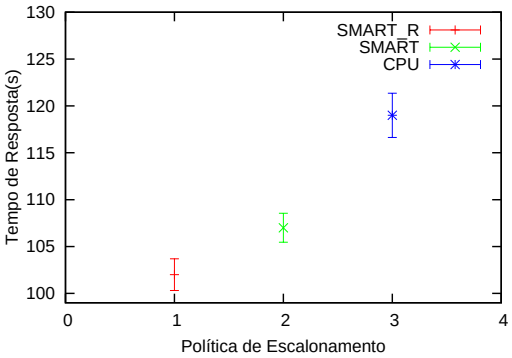
política. As figuras mostram que a política baseada no algoritmo de alerta produz o menor tempo de execução para a aplicação MapReduce. Mais uma vez, o algoritmo de alerta se mostrou melhor que um classificador isolado.

Figura 5.4: Desempenho da aplicação MapReduce usando o algoritmo de alerta de problemas de desempenho.

(a) Tempo de Resposta



(b) Intervalo de Confiança



5.2 Diagnóstico de Problemas de Desempenho

Nesta seção, testes estatísticos são usados para construir mecanismos de diagnóstico de problemas de desempenho. Em muitos casos, quando um problema de desempenho é estimado, a determinação das causas do problema detectado pode facilitar uma ação de reparo. Mecanismos de diagnóstico são importantes para a caracterização completa de um problema de desempenho.

Alguns algoritmos de aprendizado de máquina, como redes Bayesianas e árvores de decisão, permitem interpretar, de forma simples, as métricas que exercem maior influência em um processo decisório. Todavia, essa propriedade não é geral e muitos algoritmos de aprendizado de máquina não possuem tal característica. De fato, estruturas de classificadores como SVM e redes neurais podem ser de difícil interpretação.

Uma abordagem alternativa para contornar essa limitação dos algoritmos de aprendizado é buscar mecanismos de diagnóstico independentes do processo de estimação ou previsão. Testes estatísticos podem ser usados com esse intuito. Tipicamente, testes estatísticos são usados para estimar as diferenças entre duas populações de forma simples e eficiente (NIST/SEMATECH, 2011). Particularmente, neste trabalho, testes estatísticos são aplicados para aprender a diferença entre uma população em estado de violação de SLO e outra em estado de conformidade. Na próxima seção, são introduzidos três

testes estatísticos, os quais são avaliados com o intuito de quantificar o nível de influência de cada métrica observada em um problema de desempenho particular.

5.2.1 Testes Estatísticos Avaliados

Nesta seção, é analisada a aplicabilidade de três testes estatísticos para implementar um mecanismo de diagnóstico: teste de *arranjo reverso*, *z-scores* e o teste *chi-square*. Os dois primeiros testes são aplicados com o intuito de aprender as diferenças entre populações em violação e em conformidade com um objetivo. O terceiro teste, no entanto, é aplicado para estimar o grau de independência entre dois atributos, ou seja, um atributo comum e a variável de classe.

Na arquitetura de gerenciamento proposta neste trabalho, ou seja, no SMART, assume-se que o algoritmo de diagnóstico é ativado quando o algoritmo de alerta estima um período de degradação de desempenho. Nesse momento, os exemplos mais recentes usados pelo processo de estimação estão na memória. Tais exemplos serão denominados *dataset de alerta*. Um *dataset de referência* também é mantido em memória, sendo esse tomado de uma população em conformidade com o SLO definido. É importante observar que, por ser formado pelas últimas amostras coletadas, o *dataset* de alerta sofre alterações constantemente. O *dataset* de referência, ao contrário, não muda e é mantido constante na memória.

Arranjo reverso

Para aplicar o teste arranjo reverso no contexto de diagnóstico, o *dataset* de alerta é usado e comparado com o *dataset* de referência. Para cada atributo, aplica-se o teste em ambos os *datasets*, isto é, o de alerta e o de referência. Assumindo que um padrão crescente de valores para um atributo é um indicativo de degradação de desempenho, os atributos que correlacionam mais com um estado de violação são aqueles que apresentam uma tendência de crescimento significante no *dataset* de alerta e nenhuma tendência para o *dataset* de referência. Tomando a razão entre os números de arranjos reversos nos *datasets* de referência e de alerta, é possível classificar os atributos. Quanto maior for a razão, maior será o grau de correlação do atributo com uma condição de violação.

z-scores

No teste *z-scores*, o *dataset* de alerta também é comparado com o *dataset* de referência. Para cada atributo, computa-se:

$$z = \frac{m_v - m_c}{\sqrt{\frac{\sigma_v^2}{n_v} + \frac{\sigma_c^2}{n_c}}}, \tag{5-6}$$

onde m_v, σ_v^2 correspondem à média e a variância do atributo no *dataset* de alerta. m_c e σ_c^2 são a média e a variância do atributo no *dataset* de referência. n_v é o número de exemplos no *dataset* de alerta e n_c é o número de exemplos no *dataset* de referência. O teste z-scores compara o valor médio de cada atributo em ambos os *datasets*. Um valor alto e positivo de z indica que existe uma diferença significativa no valor médio do atributo nas duas populações e que o atributo é mais intenso na população em violação.

Chi-square

Para usar o teste *chi-square* com o propósito de diagnóstico, os *datasets* de alerta e de referência são combinados e o teste é aplicado no *dataset* resultante. O teste *chi-square* avalia atributos individualmente, mensurando a estatística *chi-square* em relação à variável de classe. Através de uma tabela de contingência, isto é, uma tabela de frequência onde um exemplo é classificado de acordo com dois atributos diferentes (x e y, sendo y a variável de classe), a estatística *chi-square* é computada como:

$$\chi^2 = \sum_{i,j} \frac{(f_{ij} - N \times \pi_{ij})^2}{N \times \pi_{ij}}. \tag{5-7}$$

Na Equação 5-7, f_{ij} e $N \times \pi_{ij}$ representam, respectivamente, as frequências observada e esperada na célula i, j da tabela de contingência. N é o número de exemplos e π_{ij} é a probabilidade, na população, que um indivíduo selecionado aleatoriamente apresente os valores x_i e y_j para os atributos x e y, considerando uma distribuição chi-square. Quanto maior o valor da estatística *chi-square*, maior o grau de correlação entre o atributo (x) e a variável de classe (y).

5.2.2 Experimentos

Os testes estatísticos descritos na seção anterior foram implementados no SMART para prover um mecanismo de diagnóstico de problemas de desempenho. Os testes arranjo reverso e *z-scores* foram implementados em Java

e o teste chi-square usou a implementação disponível na ferramenta Weka. A seguir, descrevemos os *datasets* utilizados nos experimentos.

Geração dos Datasets

Para avaliar a eficácia dos testes estatísticos para diagnosticar problemas de desempenho, foram coletados dados da execução da aplicação *WordCount* em diferentes cenários de carga. Dois *datasets* foram criados, cada um representando 12.5 minutos de execução. Os dados foram coletados a cada 15 segundos.

Cada *dataset* é criado em um cenário que abrange três componentes consecutivos de carga, $\langle c_1, c_2, c_3 \rangle$. O primeiro componente de carga, c_1, executa durante 300 segundos; o segundo e o terceiro componentes, c_2 e c_3, executam, cada um, por 225 segundos. Cada componente de carga, por sua vez, é uma tupla do tipo c_k=i_c:i_d:i_n, onde i_c, i_d e i_n representam, respectivamente, uma intensidade de carga de CPU, disco e rede. De acordo com essa notação, um cenário como ⟨20:00:00,50:00:00,80:00:00⟩ representa uma execução da aplicação MapReduce onde os primeiros 300 segundos são marcados por um padrão de carga de 20% de CPU; os próximos 225 segundos são marcados por um padrão de carga de 50% de CPU e os últimos 225 segundos de execução demonstram um padrão de carga de 80% de CPU. Disco e rede não são exercitados nesse cenário.

Os dois *datasets* gerados e os cenários de carga que os originou são mostrados nas duas primeiras linhas da Tabela 5.5. A última linha dessa tabela representa o *dataset* disponível quando o algoritmo de alerta de problemas de desempenho sinaliza uma degradação no experimento que utiliza a carga de CPU em escada (Seção 5.1.2). Esse *dataset* consiste dos 50 últimos exemplos coletados antes do algoritmo ativar o alarme.

Nos experimentos a seguir, os *datasets* mostrados na Tabela 5.5 são usados como *datasets* de alerta. Por outro lado, o *dataset* de referência abrange 50 exemplos retirados de um *log* de execução da aplicação *WordCount*, quando o sistema não estava sob o efeito de injetores de carga.

Tabela 5.5: *Datasets* para avaliação dos mecanismos de diagnóstico.

Dataset	Cenário
disk	⟨00:20:00,00:50:00,00:80:00⟩
cpu-net	⟨20:00:20,50:00:50,50:00:80⟩
cpu-step	Carga cpu-step

Resultados

Para determinar o quão acurado é um teste estatístico ao determinar as métricas de maior influência sobre um problema particular, a métrica S_{rank} foi proposta. Essa métrica se baseia em dois conjuntos: *Rank* e *Stress*. *Rank* é o conjunto formado pelos três primeiros atributos classificados pelo teste estatístico como os de maior influência para o problema de desempenho percebido em um experimento. *Stress* é o conjunto formado pelos recursos (CPU, disco, rede, etc) que receberam algum tipo de injeção de carga durante o experimento. A Equação 5-8 calcula a pontuação acumulada por um teste estatístico:

$$S_{rank} = \sum_{i=1}^{3}(4-i) * h_i. \tag{5-8}$$

Na Equação 5-8, o termo $(4-i)$, $i = 1, .., 3$, representa a recompensa associada ao i-ésimo atributo de *Rank*. h_i é uma função indicadora que resulta em 0, se o i-ésimo atributo de *Rank* não está associado a nenhum recurso de *Stress*. Caso contrário, a função resulta no valor 1.

Cada teste estatístico foi avaliado sobre os três *datasets* da Tabela 5.5. A Tabela 5.6 demonstra os resultados obtidos. Para o *dataset disk*, por exemplo, tem-se os seguintes conjuntos:

$$\begin{aligned}
\text{Stress} &= \{disco\} \\
Rank^1 &= \{memory_usage, disk_bytes_read, net_bytes_in\} \\
Rank^2 &= \{disk_bytes_write, disk_bytes_read, nfs_bytes_write\} \\
Rank^3 &= \{disk_bytes_read, disk_bytes_write, net_bytes_in\}.
\end{aligned}$$

$Rank^1$, $Rank^2$ e $Rank^3$ correspondem, respectivamente, ao conjunto *Rank* dos testes arranjo reverso, z-scores e chi-square. Esses conjuntos resultam nas seguintes pontuações:

$$\begin{aligned}
{S_{Rank}}^1 &= 3 \times 0 + 2 + 1 \times 0 = 2 \\
{S_{Rank}}^2 &= 3 \times 1 + 2 + 1 \times 0 = 5 \\
{S_{Rank}}^3 &= 3 \times 1 + 2 + 1 \times 0 = 5.
\end{aligned}$$

${S_{Rank}}^1$, ${S_{Rank}}^2$ e ${S_{Rank}}^3$ correspondem ao conjunto S_{Rank} dos testes arranjo reverso, z-scores e chi-square, respectivamente. É possível notar que o teste arranjo reverso classificou incorretamente a métrica *memory_usage* como aquela mais relacionada a uma violação. A segunda métrica, *disk_bytes_read*,

está diretamente ligada à utilização de disco, porém a terceira métrica não. Portanto, considerando o *dataset disk*, a pontuação obtida pelo teste arranjo reverso foi 2. Essa pontuação é menor que as obtidas pelos outros dois testes estatísticos, uma vez que as duas primeiras métricas selecionadas por aqueles testes estão diretamente relacionadas à utilização de disco.

Tabela 5.6: Métricas selecionadas pelos testes estatísticos

	reverse arrangements		z-scores		chi-square	
Dataset	Atributo	S_{rank}	Atributo	S_{rank}	Atributo	S_{rank}
disk	memory_usage disk_bytes_read net_bytes_in	2	disk_bytes_write disk_bytes_read nfs_bytes_write	5	disk_bytes_read disk_bytes_write net_bytes_in	5
cpu-net	memory_usage cpu_usage net_bytes_in	3	net_bytes_out cpu_usage disk_bytes_read	5	net_bytes_in cpu_usage net_bytes_out	6
cpu-step	net_bytes_in net_bytes_out nfs_bytes_read	0	cpu_usage containers_avg net_bytes_in	5	disk_bytes_write nfs_bytes_write cpu_usage	1

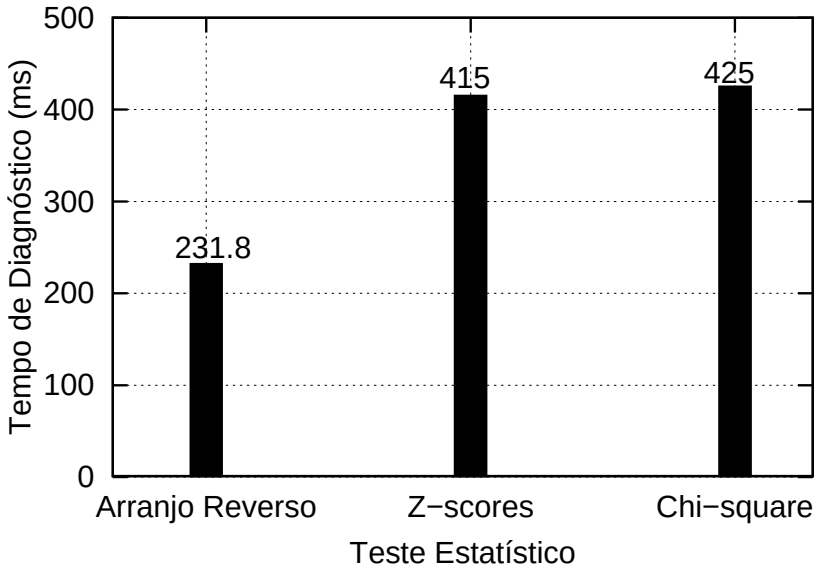
De maneira geral, considerando os dois primeiros *datasets* da Tabela 5.5, o teste *chi-square* atingiu a maior pontuação, especialmente naqueles *datasets* envolvendo combinações de carga. O teste *z-scores* atingiu a segunda maior pontuação, seguido pelo teste arranjo reverso. Todavia, o experimento utilizando o terceiro *dataset*, *cpu-step*, mostrou uma situação diferente. Nesse caso particular, o desempenho do teste *chi-square* foi muito inferior ao *z-score*. A justificativa para tão baixo desempenho é o fato do *dataset* em questão possuir poucas instâncias (exemplos) em estado de violação. Como o *dataset cpu-step* abrange os últimos exemplos antes do algoritmo de alerta ativar o alarme de degradação de desempenho, há, nesse *dataset*, uma preponderância de exemplos em estado de não-violação. Por outro lado, o teste *z-scores* se mostrou menos sensíveis ao número de exemplos em estado de violação. Como observado nos experimentos anteriores, o teste *z-scores* atingiu uma pontuação superior ao teste arranjo reverso. Acreditamos que a informação temporal requerida pelo teste arranjo reverso não foi importante para diagnosticar problemas de desempenho. Para confirmar essa suspeita, no futuro, é necessário investigar outros testes estatísticos que não façam suposições de ordem temporal.

Finalmente, para avaliar a eficiência dos testes estatísticos, o tempo requerido pelos algoritmos para diagnosticar um problema de desempenho foi avaliado. A Figura 5.5 ilustra esses tempos. É possível notar que os testes arranjo reverso, *z-scores* e *chi-square* demandam, respectivamente, 231, 415 e 425 milissegundos para diagnosticar um problema. Esses valores representam a média dos tempos obtidos nos três *datasets* considerados. Podemos notar que

todos os testes levam menos de 1 segundo para realizar um diagnóstico. Esse resultado confirma a eficiência dos testes estatísticos.

Basicamente, foi observado que o teste *z-scores* apresenta acurácia satisfatória para diagnosticar problemas de desempenho. Esse comportamento foi observado tanto em *datasets* envolvendo combinações de carga, como aqueles com poucas instâncias em estado de violação. Além disso, o teste se mostrou muito eficiente.

Figura 5.5: Tempo de resposta requerido por cada teste estatístico para diagnosticar um problema.



5.3 Considerações Finais

Neste capítulo, foi apresentado um algoritmo de alerta de problemas de desempenho que aumenta o poder de estimação dos classificadores pela atenuação da emissão de alarmes falsos. Foram realizados vários experimentos com o intuito de avaliar o desempenho do algoritmo. Em todos os cenários avaliados, o algoritmo proposto atenuou os efeitos de alarmes falsos e conseguiu um desempenho superior àquele apresentado pelos classificadores puros. Esse resultado pode ser explicado pelo fato do algoritmo levar em consideração a sequência temporal das amostras, isto é, uma amostra não é classificada isoladamente, mas sim levando em consideração amostras anteriores. O peso que se atribui ao passado, no entanto, pode tornar o algoritmo mais reativo. De certa forma, os parâmetros $thUpper$, $thLower$ e $thRag$ podem ser usados para controlar esse compromisso entre robustez e acurácia do algoritmo proposto.

Porém, uma limitação desse algoritmo é que a parametrização dessas variáveis dependente do *dataset* usado.

O uso de testes estatísticos também foi proposto para classificar o nível de influência de cada métrica observada em um problema de desempenho particular. Três testes estatísticos foram avaliados utilizando diferentes *datasets*. Particularmente, para um mecanismo *online* de diagnóstico, o teste *z-scores* mostrou-se bastante adequado. Comparando o quanto duas populações são diferentes estatisticamente, o teste *z-scores* conseguiu capturar o padrão de crescimento das variáveis relacionadas à utilização de recursos e classificar os atributos de forma eficiente e eficaz de acordo com este padrão. Comparado a técnicas tradicionais de aprendizado de máquina que tentam inferir o grau de dependência de duas variáveis, o teste *z-scores* tem a seu favor a simplicidade, uma vez que o teste estatístico dispensa a necessidade de treinamento prévio e não necessita de nenhuma parametrização. No entanto, vale ressaltar que o teste *z-scores* se baseia em estatística Gaussiana e pode ser menos aplicável em dados não paramétricos, ou seja, aqueles que não se aproximam de distribuições conhecidas.

No próximo capítulo, é mostrada a avaliação de um algoritmo de classificação *online*. O intuito é obter um mecanismo de previsão de problemas de desempenho onde o processo de aprendizado possa acontecer de forma *online* e incremental.