DASRS Rest: Um Algoritmo Eficiente de Detecção de Anomalias em Tempo Real para Data Centers

Ricardo Dias[1,2], **Leopoldo A. F. Mauricio**[2], **Marcus Poggi**[1]

[1]Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio - Brasil
[2]Globo.com - Brasil

{rdias,poggi}@inf.puc-rio.br, {ricardo.dias,leopoldo}@g.globo

Abstract. *A large amount of equipment, with different configurations, and regular software and hardware upgrades on data centers, make it challenging to use monitoring systems based on threshold setting. This paper proposes to use anomaly detection for fault prevention. It presents the Decreased Anomaly Score by Repeated Sequence (DASRS) Rest algorithm, which detects anomalies without requiring prior knowledge of the monitored service. We evaluate DASRS Rest performance using the Numenta Anomaly Benchmark (NAB) framework. The proposed algorithm has good accuracy results, the lowest memory consumption, and is the fastest when compared to several state-of-the-art algorithms.*

Resumo. *A grande quantidade de equipamentos, com diferentes configurações, e atualizações constantes de software e hardware em data centers, tornam difícil o uso de sistemas de monitoração baseados na configuração de thresholds. Este artigo propõe utilizar detecção de anomalias para prevenção de falhas e apresenta o algoritmo Decreased Anomaly Score by Repeated Sequence (DASRS) Rest, que detecta anomalias sem exigir conhecimento prévio do serviço monitorado. Avaliamos o desempenho do DASRS Rest utilizando o framework Numenta Anomaly Benchmark (NAB). O algoritmo proposto possui bons resultados de acurácia, o menor consumo de memória e é o mais rápido, quando comparado com diversos algoritmos do estado da arte.*

1. Introdução

Data centers usualmente hospedam milhares de equipamentos, onde múltiplas aplicações são executadas [Couto et al. 2012]. São ambientes dinâmicos, no qual a atualização de software e hardware é frequente e as cargas de trabalho das aplicações variam em função da sazonalidade [Claps et al. 2015]. A interrupção dos serviços hospedados pode gerar prejuízos financeiros, por isso, os equipamentos e aplicações são geralmente monitorados. Os sistemas de monitoração coletam métricas de equipamentos e aplicações tais como: utilização de CPU e memória, volume de tráfego de rede, número de sessões de um banco de dados, entre outras.

Entretanto, muitas vezes a detecção de falhas ainda depende da configuração prévia de limiares[1] estáticos das métricas monitoradas. Nesse tipo de solução, é difícil

Este trabalho foi realizado com recursos do CNPq, CAPES e Globo.com.

[1]*thresholds*

definir e manter limiares adequados porque o ambiente é dinâmico. Seria necessário demandar muito tempo dos administradores de sistemas, que precisariam conhecer profundamente o comportamento de cada serviço monitorado [Gabel et al. 2012]. Um limiar ótimo precisa ser suficientemente baixo para que falhas não deixem de ser identificadas e suficientemente alto para não gerar falsos alarmes. Além disso, os limiares precisam variar para acompanhar a sazonalidade das cargas de trabalho dos servidores bem como atualizações de software ou hardware. Na prática, é comum encontrar limiares configurados com valores muitos altos nos data centers de produção. Por isso, costumeiramente os alarmes são disparados quando os serviços já estão indisponíveis, causando prejuízos financeiros. Além disso, a ação dos administradores de sistema é, frequentemente, apenas reativa, realizada quando a indisponibilidade indesejada já aconteceu.

Este artigo propõe o uso de técnicas de detecção de anomalias, que não exijam conhecimento prévio sobre o serviço monitorado, para prevenção de falhas. A detecção de anomalias em data centers deve ser realizada a partir da análise em tempo real das métricas coletadas dos servidores e aplicações. O algoritmo de detecção de anomalia deve possuir boa acurácia e capacidade para se adaptar automaticamente às possíveis mudanças no padrão das métricas monitoradas. O volume de métricas coletados em um data center é geralmente muito grande. Por esse motivo, os algoritmos de detecção de anomalias em tempo real devem ser eficientes, apresentando baixa complexidade, pouco consumo de memória e tempo de execução reduzido.

Apresentamos um novo algoritmo de detecção de anomalia chamado DASRS (*Decreased Anomaly Score by Repeated Sequence*) Rest. Seu objetivo é detectar anomalias, a partir da análise em tempo real das métricas coletadas de servidores e aplicações de data centers de produção. As métricas coletadas se traduzem em séries temporais [Brockwell and Davis 2016] analisadas pelo DASRS Rest, que identifica os trechos com comportamento incomum, diferente do comportamento normal anterior. Criamos um sistema de pontuação para classificar um comportamento incomum encontrado. Nele, cada possível anomalia identificada recebe uma pontuação que indica a probabilidade de ser realmente uma anomalia. Além disso, a pontuação de um comportamento incomum que se repete é automaticamente diminuída. Assim, um novo padrão de comportamento deixa de ser classificado como anomalia quando se torna frequente. Dessa forma, as variações de carga de trabalho de servidores, que podem ocorrer em função da sazonalidade, deixam de ser consideradas como anomalias. O DASRS Rest possui bons resultados de acurácia (pontuação), o menor consumo de memória e é o mais rápido, quando comparado com diversos algoritmos do estado da arte encontrados no *framework* NAB (*Numenta Anomaly Benchmark*) [Lavin and Ahmad 2015].

O restante do artigo está organizado da seguinte forma. Os trabalhos relacionados estão na Seção 2. O algoritmo de detecção de anomalias proposto é discutido na Seção 3. O ambiente de testes desenvolvido é detalhado na Seção 4. Os resultados experimentais são apresentados na Seção 5. A Seção 6 apresenta a conclusão e os trabalhos futuros.

2. Trabalhos Relacionados

A detecção de anomalias em séries temporais é uma área bastante estudada desde que o tema foi apresentado por Fox [Fox 1972]. Adams e MacKay [Adams and MacKay 2007] propõem um modelo de ponto de mudança alternativo. Em lugar de analisar os pontos de

mudança retrospectivamente, o algoritmo é interessado na mudança mais recente. Chandola et al. [Chandola et al. 2009] propõem o agrupamento das técnicas de detecção de anomalias em diferentes categorias, em função da abordagem central de cada técnica. Wang et al. [Wang et al. 2011] propõem técnicas de detecção de anomalias baseadas nas estatísticas Tukey e Entropia Relativa, enquanto Schneider et al. [Schneider et al. 2016] implementa o algoritmo *EXPected Similarity Estimation* (EXPoSE), baseado em *kernel* e capaz de calcular eficientemente a similaridade entre novos pontos de dados e a distribuição de dados regulares.

Skyline [Etsy 2013] utiliza um conjunto de técnicas estatísticas e um esquema de votação para gerar a pontuação final da anomalia detectada. As técnicas estatísticas incluem desvio absoluto mediano, média da primeira hora, desvio padrão da média, desvio padrão da média móvel, subtração da média acumulada, mínimos quadrados e caixas de histograma. Os testes estatísticos são realizados apenas com as últimas observações analisadas. O algoritmo Earthgecko Skyline [Earthgecko 2019] aprimora o Skyline com técnicas que aumentam o desempenho do algoritmo e otimizam a lógica do esquema de votação, enquanto Kejariwal [Kejariwal 2015] combina o teste *Extreme Studentized Deviate* (ESD) generalizado [Rosner 1983] com as métricas estatísticas a fim de que a aproximação por partes seja utilizada para detectar tendências de longo prazo. O algoritmo foi projetado especificamente para detectar anomalias sazonais no contexto de dados de redes sociais e funciona bem quando surgem anomalias em dados periódicos que não são muito diferentes dos dados anteriores.

Smirnov [Smirnov 2016] desenvolveu o algoritmo *Contextual Anomaly Detector* (CAD) que gera uma pontuação de anomalia a cada observação de entrada indicando quanto o valor observado corresponde ou não a um novo contexto em função das observações precedentes. Burnaev e Ishimtsev [Burnaev and Ishimtsev 2016] propõem algoritmos de detecção de anomalias baseados em distância e densidade. Os algoritmos executam métodos de extração de características (*features*). Em seguida, definem uma pontuação quando o novo valor difere significativamente dos dados previamente observados. A partir daí, os algoritmos realizam uma interpretação probabilística da pontuação. [Guha et al. 2016] propõe o método *Robust Random Cut Forest* (RRCT), que realiza cortes aleatórios de estruturas de dados em árvores. A anomalia é baseada na influência de uma observação ainda não vista sobre o restante dos dados.

Em [Ahmad and Purdy 2016, Ahmad et al. 2017] os autores implementam o Numenta *Hierarchical Temporal Memory* (HTM) (ou Numenta), onde a técnica de detecção de anomalia é baseada na memória temporal hierárquica. As sequencias temporais de dados são modeladas e, para cada momento t, o algoritmo realiza múltiplas previsões para o valor do dado no momento $t + 1$. Tais previsões são então comparadas com o valor real obtido a fim de determinar se ele pode ser considerado normal ou anômalo. O *framework Numenta Anomaly Benchmark* (NAB) é proposto em [Lavin and Ahmad 2015] para comparar algoritmos de detecção de anomalias em séries temporais. Um conjunto de ferramentas permite testar, medir e comparar diferentes algoritmos de detecção de anomalias. Por ser um *framework* utilizado em diversas pesquisas [Ahmad et al. 2017, Burnaev and Ishimtsev 2016, Munir et al. 2018, Safin and Burnaev 2017], utilizamos o NAB para avaliar o desempenho do algoritmo DASRS Rest.

3. O Algoritmo Proposto

O algoritmo DASRS Rest, proposto nesse artigo, identifica e contabiliza as sequências de valores normalizados que aparecem numa série temporal e gera um escore em função da quantidade de vezes que cada sequência foi identificada. Na primeira vez que uma determinada sequência é identificada o escore retornado é o maior possível, pois o algoritmo interpreta como um comportamento novo. A partir daí, à medida que a sequência se repete o algoritmo reduz o escore automaticamente.

Algoritmo 1: DASRS Rest

Entrada:

- $minValue$: valor mínimo entre as observações
- $maxValue$: valor máximo entre as observações
- θ: quantidade de valores normalizados
- $sequenceSize$: tamanho da janela de sequencias
- $restPeriod$: período que o escore de anomalia é atenuado após ter detectado uma anomalia
- x_i: observação corrente

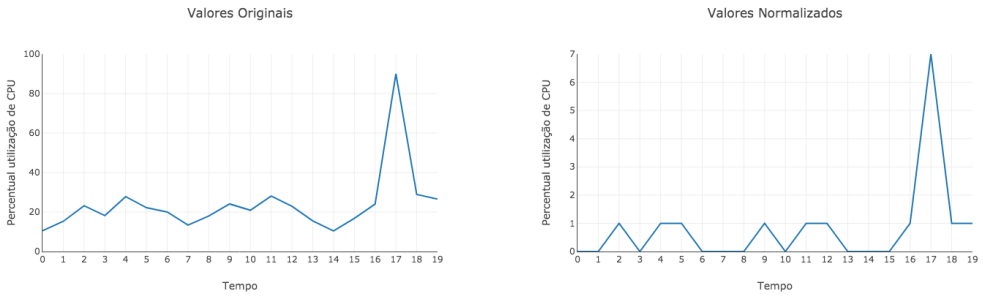
Saída: $AnomalyScore \in [0,1]$: escore da anomalia

```
1  Function init(minValue, maxValue, θ, sequenceSize, restPeriod):
2      restWeakenFactor ← 0
3      normInputSequence ← []
4      sequences ← {}
5      minVal ← null
6      maxVal ← null
7  Function getRawAnomalyScore(occurrences):
8      return 1/occurrences
9  Function getRestScore(score):
10     if restWeakenFactor > 0 then
11         score ← score/restWeakenFactor
12         restWeakenFactor ← restWeakenFactor − 1
13     else if score ≥ 1 then
14         restWeakenFactor ← restPeriod
15     return score
16 Function handleRecord(x_i):
17     normInpVal ← ⌊θ × (x_i − minValue)/(maxValue − minValue)⌋
18     normInputSequence.append(normInpVal)
19     if len(normInputSequence) < sequenceSize then
20         return 0
21     end
22     if normInputSequence ∈ sequences then
23         sequences[normInputSequence] += 1
24     else
25         sequences[normInputSequence] = 1
26     end
27     occurrences = count(normInputSequence in sequences)
28     normInputSequence.pop(0)
29     rawAnomalyScore ← getRawAnomalyScore(occurrences)
30     anomalyScore ← getRestScore(rawAnomalyScore)
31     return anomalyScore
```

Sendo X_t a série temporal formada pelas observações $x_1, x_2, \ldots$, uma sequência de X_t é um subconjunto de X_t formado por elementos consecutivos, por exemplo, $x_i, \ldots, x_j$, com $i < j$. A normalização aplicada pelo algoritmo consiste em transformar o valor da observação em um número inteiro entre 0 e um fator de normalização que iremos chamar de θ. Quanto maior o valor de θ, que deve ser um número maior ou igual a 1, maior é a quantidade de elementos possíveis em uma série, que é normalizada para se tornar computacionalmente mais leve. A equação abaixo (Algoritmo 1, Linha 17) exibe as operações realizadas em x_i para obter seu valor normalizado (x'_i):

$$x'_i = \left\lfloor \frac{x_i - min_X}{max_X - min_X} \times \theta \right\rfloor \quad (1)$$

onde x_i é a observação de entrada ($x_i \in \mathbb{R}$), min_X e max_X são, respectivamente, o menor e o maior valor atribuídos às observações da série temporal X_t, θ representa o fator de normalização e x'_i é o valor normalizado da observação de entrada x_i, $x'_i \in \mathbb{N}$ e $0 \leq x'_i \leq \theta$. Decidimos normalizar os valores observados para limitar a quantidade de sequências distintas, sem alterar as principais características da série temporal. A figura 1(a) representa uma série temporal com observações de percentual de utilização de CPU, onde o valor mínimo é 0 e o valor máximo é 100. Entretanto, a figura 1(b) mostra que é possível utilizar um θ com valor igual a 7, por exemplo, para normalizar os dados e limitar a quantidade de sequências mantendo as principais características da série temporal original.



(a) Série temporal com os valores originais (b) Série temporal com os valores normalizados

Figura 1. Exemplo de série temporal normalizada

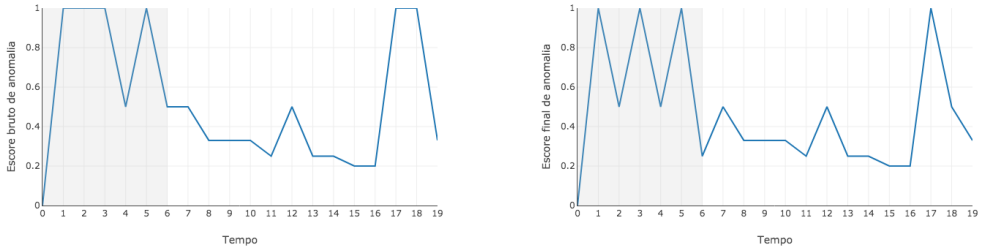
Ao calcular a pontuação de anomalia levamos em conta a sequência normalizada corrente e a quantidade de vezes que essa sequência ocorreu no passado (Algoritmo 1, Linha 27). A quantidade de elementos da sequência é definida por um parâmetro, que decidimos chamar de $sequenceSize$. A sequência normalizada corrente é formada pelos últimos $sequenceSize$ elementos normalizados da série temporal. Guardamos numa estrutura de dados a quantidade de vezes que cada sequência apareceu durante o processamento da série temporal. A equação abaixo é utilizada no cálculo do escore de anomalia (Algoritmo 1, Linha 8):

$$score = \frac{1}{occurrences} \tag{2}$$

onde $occurrences$ representa a quantidade de vezes que a sequência corrente foi encontrada. É esperado que no início do processamento de uma série temporal o algoritmo gere escores altos de anomalia. Isso ocorre porque até então as sequências foram encontradas poucas vezes pelo algoritmo. Por este motivo, deve-se considerar o tempo inicial de processamento de uma série como período de treinamento, ou período probatório, e as anomalias identificadas neste período não devem ser consideradas para disparar alarmes.

No algoritmo DASRS Rest (Algoritmo 1) o escore gerado pela equação 2 não é a pontuação final porque [Ahmad et al. 2017] mostrou que muitas séries de dados possuem comportamento imprevisível, causado por ruídos ou pela natureza aleatória de algumas métricas, gerando um grande número de falsos positivos. Para lidar com esse tipo de problema, o DASRS Rest associa as anomalias muito próximas ao mesmo fenômeno. Por isso, após identificar uma anomalia, os escores seguintes tem seu valor diminuído, por um período definido pelo parâmetro $restPeriod$. A função $getRestScore$ mostra como

o parâmetro $restPeriod$ é utilizado para atenuar o escore final de anomalia (Algoritmo 1, Linhas 9-15). As figuras 2(a) e 2(b) mostram respectivamente os escores bruto e final de anomalia gerados pelo DASRS Rest para a série temporal da figura 1(a), com $\theta = 7$, $sequenceSize = 2$ e $restPeriod = 2$. Detalhamos a escolha dos parâmetros de entrada θ, $sequenceSize$ e $restPeriod$ em [Dias 2019]. A área em cinza da figura representa o período de treinamento, quando o algoritmo aprende quais são os padrões existente na série. A tabela 1 mostra o passo a passo de execução do DASRS Rest para a série da figura 1(a). A série temporal é representada pelas colunas Tempo e X, que informam o valor escalar de uma série temporal ao longo do tempo. X' é o valor normalizado de X, Sequência são os últimos 2 ($sequenceSize$) elementos de X', Ocorrências é a quantidade de vezes que a Sequência foi encontrada, Escore Bruto é o escore bruto de anomalia calculado pelo algoritmo (Algoritmo 1, Linhas 7-8) e Escore Final é o escore retornado pelo algoritmo após atenuar o valor do escore bruto pelo período $restPeriod$ (Algoritmo 1, Linha 30).



(a) Escore bruto de anomalia (b) Escore final de anomalia

Figura 2. Escore de anomalia calculados pelo DASRS Rest

Tabela 1. Passo a passo de processamento do DASRS Rest

Tempo	X	X'	Sequência	Ocorrências	Escore Bruto	Escore Final
0	10,5	0	-	-	0	0
1	15,3	0	(0, 0)	1	1	1
2	23,2	1	(0, 1)	1	1	0,5
3	18,2	0	(1, 0)	1	1	1
4	27,8	1	(0, 1)	2	0,5	0,5
5	22,2	1	(1, 1)	1	1	1
6	20,0	0	(1, 0)	2	0,5	0,25
7	13,4	0	(0, 0)	2	0,5	0,5
8	19,0	0	(0, 0)	3	0,33	0,33
9	24,1	1	(0, 1)	3	0,33	0,33
10	20,9	0	(1, 0)	3	0,33	0,33
11	28,1	1	(0, 1)	4	0,25	0,25
12	22,9	1	(1, 1)	2	0,5	0,5
13	15,5	0	(1, 0)	4	0,25	0,25
14	10,4	0	(0, 0)	4	0,25	0,25
15	16,8	0	(0, 0)	5	0,2	0,2
16	24,0	1	(0, 1)	5	0,2	0,2
17	90,0	7	(1, 7)	1	1	1
18	28,9	1	(7, 1)	1	1	0,5
19	26,6	1	(1, 1)	3	0,33	0,33

4. O Ambiente de Desenvolvimento e Testes

Avaliamos o DASRS Rest utilizando o *framework* NAB, que fornece um conjunto de ferramentas que permitem comparar a acurácia de diferentes algoritmos de detecção de anomalias. O NAB[2] viabiliza o processamento de *streaming* de dados em um ambiente controlado e repetível, além de disponibilizar seu código, diversos algoritmos de detecção

[2]https://github.com/numenta/NAB

de anomalia do estado da arte e suas documentações. Nele, também é possível encontrar o histórico dos resultados da análise de séries temporais processadas por diferentes algoritmos de detecção de anomalia do estado da arte.

4.1. Cálculo da Pontuação de Algoritmos de Detecção Utilizando o *Framework* NAB

No *framework* NAB os algoritmos de detecção são recompensados quando identificam uma anomalia de forma correta e são penalizados pelas anomalias não detectadas ou assinaladas de forma incorreta. Quanto mais cedo a anomalia é detectada, maior é a pontuação atribuída ao algoritmo. Além disso, janelas de anomalia são utilizadas para atribuir pontos às detecções corretas que são rapidamente realizadas. As janelas representam um intervalo de pontos de dados que é centralizado em torno de um rótulo de anomalia verdadeira. A implementação das janelas de anomalia, dos cálculos de pontuação, entre outros, no *framework* NAB, estão detalhados em [Lavin and Ahmad 2015, Singh and Olinsky 2017]. Uma função de pontuação é aplicada em cada janela de anomalia para identificar e avaliar as detecções realizadas de forma correta (*True Positive* - TP), as detecções incorretas (*False Positive* - FP) e as anomalias não detectadas (*False Negative* - FN). As detecções em uma janela identificam corretamente dados anômalos e aumentam a pontuação no NAB. Se ocorrer uma detecção no início de uma janela, será dado maior valor pela função de pontuação do que uma detecção no final da janela. Detecções múltiplas dentro de uma única janela identificam os mesmos dados anômalos, portanto, é usado somente a detecção mais antiga (mais valiosa) para a contribuição da pontuação. As detecções que estão fora das janelas são falsos positivos (FP), dando uma contribuição negativa para a pontuação do NAB. O local onde o FP é identificado também é usado na função de pontuação, de modo que um FP que ocorre logo após uma janela prejudica menos a pontuação do NAB que um FP que ocorre longe da janela. Perder uma janela completamente, ou um falso negativo (FN), resulta em uma contribuição negativa para a pontuação. Os pontos fora das janelas que não tiveram anomalias identificados, ou seja, verdadeiro negativo (*True Negative* - TN) não contribuem para a pontuação.

Através de perfis de aplicação é possível definir pesos aos falsos positivos e falsos negativos, a fim de atender cenários em que menos detecções perdidas ou menos detecções erradas são mais valorizadas. Três diferentes perfis de aplicação são utilizados: padrão, recompensa poucos FPs e recompensa poucos FNs. O perfil padrão atribui TPs, FPs, TNs e FNs com pesos relativos, de forma que as detecções aleatórias feitas em 10% do tempo obtém uma pontuação final zero em média. Os dois últimos perfis definem penalidades maiores para FPs e FNs, respectivamente. Através deles, é possível ajustar os pesos das penalidades de acordo com as necessidades específicas da aplicação. A tabela 2 mostra os pesos propostos pelo NAB para TP, FP, TN e FN para cada um dos perfis.

Tabela 2. Perfis de aplicação

	Pesos			
Perfil	TP	FP	TN	FN
Padrão	1.0	-0.11	1.0	-1.0
Recompensa Poucos FPs	1.0	-0.21	1.0	-1.0
Recompensa Poucos FNs	1.0	-0.11	1.0	-2.0

A pontuação final para um determinado algoritmo é calculada da seguinte maneira. Seja A um perfil de aplicação e $A_{TP}, A_{FP}, A_{FN}, A_{TN}$ os pesos correspondentes a TP, FP, FN, TN. Seja D o conjunto de arquivos de dados e Y_d o conjunto de instâncias de anomalias detectadas em d. O número de janelas com zero detecções

neste arquivo é o número de falsos negativos, representados por f_d. A função sigmoidal, $\sigma^A(y) = (A_{TP} - A_{FP})\left(\frac{1}{1+e^{5y}}\right) - 1$, define o peso das detecções individuais, dada uma janela de anomalia e a posição relativa de cada detecção, onde y é a posição relativa da detecção dentro da janela de anomalia. Faltar completamente uma janela é contado como um falso negativo e atribuiu uma pontuação de A_{FN}. A pontuação de um arquivo de dados, $S_d^A = \left(\sum_{y \in Y_d} \sigma^A(y)\right) + A_{FN} f_d$, é a soma das pontuações das detecções individuais mais o peso das janelas que não tiveram anomalias detectadas. Esta pontuação, portanto, acumula a pontuação ponderada de cada verdadeiro positivo e falso positivo, abatendo os falsos negativos com uma contagem ponderada. A pontuação bruta de referência para um determinado algoritmo, $S^A = \sum_{d \in D} S_d^A$, é a soma das pontuações brutas de todos os arquivos de dados. Portanto, a pontuação final, calculada a partir da equação $S_{NAB}^A = 100 \times \frac{S^A - S_{null}^A}{S_{perfect}^A - S_{null}^A}$, é uma pontuação normalizada, onde $S_{perfect}^A$ é a pontuação obtida por um detector de anomalia "perfeito" (um que produz todos os verdadeiros positivos no início da janela e nenhum falso positivo) e S_{null}^A é a pontuação obtida por um detector "nulo" (um que não produz nenhuma detecção de anomalia).

4.2. Datasets Utilizados na Avaliação dos Algoritmos de Detecção

O *framework* NAB possui um *dataset* nativo[3] formado por 58 séries temporais univariadas que possuem registro de data e hora, além de um valor escalar, em cada linha armazenada. Cada série possui entre 1.000 e 22.000 observações, totalizando 365.551 observações. O *dataset* possui diferentes tipos de métricas. Entre elas estão informações de tráfego rodoviário, utilização da CPU, de rede e métricas de mídias sociais. Alguns arquivos foram gerados artificialmente, enquanto outros foram criados a partir de dados coletados do mundo real. O NAB fornece rótulos para todas as anomalias existentes em seu *dataset* e é possível adicionar e processar *datasets* externos no *framework*.

O Yahoo! disponibiliza um *dataset* chamado Webscope que possui séries temporais e rótulos de anomalias [4]. O *dataset* possui 4 conjuntos de dados: A1Benchmark, A2Benchmark, A3Benchmark e A4Benchmark. Decidimos utilizar apenas o conjunto de dados A1Benchmark porque é o único criado a partir de métricas reais de servidores e aplicações de produção do Yahoo!, os demais foram criados a partir de dados sintéticos. Existem 67 séries temporais no A1Benchmark. Cada uma possui entre 741 e 1.461 observações, totalizando 94.866 observações. Cada série é armazenada em um arquivo e cada linha contém um *timestamp*, um valor escalar e um booleano indicando se o registro é anômalo ou não. Alguns rótulos de anomalia do *dataset* Yahoo! A1Benchmark estão muito próximos. Por isso, problemas de sobreposição de janelas de anomalia podem acontecer quando o processamento é realizado no NAB. No entanto, é possível processar o *dataset* Yahoo! A1Benchmark no framework NAB evitando esse problema. Para isso, deve-se considerar apenas o primeiro rótulo de cada conjunto de rótulos de anomalias próximas que pertencem à mesma janela de anomalia, sem considerar os rótulos seguintes. Essa abordagem é eficiente porque, apesar de ignorados quando o *dataset* é preparado para ser processado, tais rótulos de anomalias ainda são considerados pelas detecções realizadas porque estão dentro da mesma janela da anomalia identificada.

[3] https://github.com/numenta/NAB/tree/master/data

[4] https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70

5. Resultados Experimentais

Esta seção descreve os experimentos realizados para comparar o desempenho do DASRS Rest com a de outros algoritmos de detecção de anomalia existentes na literatura. O computador utilizado nos testes é um MacBook Pro, Processador 3,1 GHz Intel Core i5, Memória 8 GB 2133 MHz LPDDR3, Sistema Operacional macOS High Sierra.

A avaliação proposta é dividida em quatro etapas. Na primeira etapa o cálculo de pontuação nativo do NAB é utilizado para medir o desempenho dos algoritmos ao buscar anomalias no *dataset* NAB e no *dataset* Yahoo-A1Benchmark. Os perfis de aplicação utilizados no cálculo da pontuação final são: perfil padrão, perfil recompensando pouco FP e perfil recompensando pouco FN. Os resultados, com as pontuações finais de cada algoritmo avaliado, estão ilustrados nas tabelas 3 e 4, organizadas em função da pontuação obtida no perfil padrão. A análise dos resultados mostra que o algoritmo DASRS Rest possui a melhor pontuação quando o *dataset* Yahoo-A1Benchmark é processado. Nesse cenário, a pontuação do DASRS Rest no perfil padrão é superior a 4 pontos quando comparada à do algoritmo CAD OSE, que ficou em segundo lugar. O algoritmo DASRS Rest possui a terceira melhor pontuação quando o *dataset* NAB é processado. Nesse cenário de testes, a pontuação do DASRS Rest é cerca de 4 pontos inferior à do CAD OSE e aproximadamente 3 pontos menor que a do algoritmo Numenta. No entanto, a diferença é de mais de 8 pontos para o algoritmo earthgecko Skyline, que possui a quarta melhor pontuação.

Tabela 3. Pontuação dos algoritmos utilizando o *dataset* Yahoo-A1Benchmark

	Pontuação por Perfil de Aplicação		
Detector	**Padrão**	**Recompensa Poucos FPs**	**Recompensa Poucos FNs**
DASRS Rest	67.1	62.6	72.3
CAD OSE	62.7	57.5	67.3
KNN CAD	52.9	41.9	58.8
Windowed Gaussian	48.9	39.7	54.8
Relative Entropy	48.9	41.4	53.6
Numenta	46.2	41.5	50.3
Bayesian Changepoint	41.1	23.9	53.7
Etsy Skyline	36.5	25.9	43.5
earthgecko Skyline	35.8	34.6	38.0
EXPoSE	13.8	10.9	26.0

Tabela 4. Pontuação dos algoritmos utilizando o *dataset* NAB

	Pontuação por Perfil de Aplicação		
Detector	**Padrão**	**Recompensa Poucos FPs**	**Recompensa Poucos FNs**
CAD OSE	69.9	67.0	73.2
Numenta	69.7	62.3	74.3
DASRS Rest	66.4	60.2	70.4
earthgecko Skyline	58.2	46.2	63.8
KNN CAD	58.0	43.4	64.8
Relative Entropy	54.1	48.0	57.9
Windowed Gaussian	40.1	23.9	47.7
Etsy Skyline	35.7	27.1	44.5
Bayesian Changepoint	17.7	5.1	32.3
EXPoSE	16.4	3.5	26.9

Na segunda etapa avaliamos o desempenho dos algoritmos em função da proporção de resultados positivos (TP) com os classificados como positivos pelo algoritmo (TP + FN): ($Recall = \frac{TP}{TP+FN}$); da proporção de resultados positivos (TP) com os que são realmente positivos (TP + FP): ($Precision = \frac{TP}{TP+FP}$) e da média harmônica entre *Precision* e *Recall*, denominada *F1-Score*: ($F1 = 2 * \frac{Precision*Recall}{Precision+Recall}$). TP é o total de anomalias detectadas corretamente, considerando apenas a primeira identificada dentro de cada janela de anomalia. FN é o total de anomalias não detectadas e FP o total de anomalias incorretamente identificadas. As anomalias detectadas no período probatório são ignoradas no cálculo do valor do FP. A escolha de qual métrica deve ser utilizada ao

comparar detectores de anomalia depende dos requisitos da aplicação. A melhor métrica de comparação será a *Recall*, por exemplo, quando é essencial encontrar o maior número possível de anomalias verdadeiras (TP) independente da quantidade de anomalias (FP) identificadas. Por outro lado, a métrica *Precision* é a melhor quando é um requisito da aplicação não detectar anomalias de forma incorreta (FP), mesmo que a detecção de anomalias verdadeiras (TP) seja prejudicada. Para as aplicações em que todas as medidas (TP, FP e FN) importam é melhor utilizar o *F1-Score* ou a pontuação nativa do NAB. Também é recomendável utilizar a pontuação do NAB quando se deseja premiar os algoritmos que mais rapidamente identificam uma anomalia. Ou quando é preciso permitir o ajuste dos pesos dos perfis de aplicação de acordo com as necessidades específicas de cada aplicação.

Os resultados estão ilustrados nas tabelas 5 e 6, que exibem os valores de TP, FP, FN, *Precision*, *Recall* e *F1-Score* quando os *datasets* NAB e Yahoo-A1Benchmark são processados. As tabelas estão ordenadas em função da pontuação obtida no parâmetro *F1-Score*. Os algoritmos com os melhores resultados são os mesmos. O DASRS Rest voltou a alcançar a melhor pontuação quando usamos o *dataset* Yahoo-A1Benchmark e o algoritmo CAD OSE foi o melhor quando utilizamos o *dataset* NAB. O parâmetro *Precision* não considera os FNs. Por isso, as anomalias não identificadas pelos algoritmos não reduziram o escore desse parâmetro. Por outro lado, o *Recall* não leva em consideração os FPs. Portanto, os algoritmos que identificam incorretamente muitas anomalias não são prejudicados no escore desse parâmetro. Os resultados do parâmetro *F1-Score* são os que mais se aproximam dos obtidos quando a pontuação nativa do NAB é utilizada porque considera os TPs, os FPs e os FNs.

Tabela 5. Benchmark utilizando o *dataset* Yahoo-A1Benchmark

Detector	TP	FN	FP	Precision	Recall	F1-Score
DASRS Rest	114	30	123	0.48	0.79	0.60
CAD OSE	110	34	160	0.41	0.76	0.53
earthgecko Skyline	61	83	33	0.65	0.42	0.51
Numenta	83	61	159	0.34	0.58	0.43
Relative Entropy	91	53	212	0.30	0.63	0.41
KNN CAD	99	45	290	0.25	0.69	0.37
Windowed Gaussian	95	49	363	0.21	0.66	0.32
Etsy Skyline	83	61	474	0.15	0.58	0.24
Bayesian Changepoint	109	35	737	0.13	0.76	0.22
EXPoSE	35	109	223	0.14	0.24	0.17

Tabela 6. Benchmark utilizando o *dataset* NAB

Detector	TP	FN	FP	Precision	Recall	F1-Score
CAD OSE	92	24	64	0.59	0.79	0.68
DASRS Rest	91	25	135	0.40	0.78	0.53
Numenta	97	19	177	0.35	0.84	0.50
Relative Entropy	76	40	133	0.36	0.66	0.47
earthgecko Skyline	87	29	265	0.25	0.75	0.37
KNN CAD	91	25	314	0.22	0.78	0.35
Windowed Gaussian	73	43	399	0.15	0.63	0.25
Etsy Skyline	72	44	497	0.13	0.62	0.21
EXPoSE	51	65	504	0.09	0.44	0.15
Bayesian Changepoint	67	49	766	0.08	0.58	0.14

Na terceira etapa avaliamos o tempo que cada algoritmo levou para processar os *datasets* NAB e Yahoo-A1Benchmark. As médias de 5 rodadas de testes estão nas tabelas 7 e 8. O tempo de execução é medido em segundos. Os resultados mostram que o algoritmo DASRS Rest é o mais rápido nos dois cenários testados. Em seguida, utilizamos uma série temporal com 10.000 observações e medimos o tempo de processamento de cada algoritmo a cada 1.000 observações analisadas. Os resultados ilustrados na figura 4 mostram que o DASRS Rest novamente obteve o melhor resultado. Além disso,

o DASRS Rest foi 3 vezes mais rápido que o melhor algoritmo do estado da arte encontrado no *framework* NAB. O resultado do Etsy Skyline foi muito ruim porque precisou de 821,34 segundos para processar as 10.000 observações. Os resultados do CAD OSE e do Numenta também foram ruins, 337,09 e 123,57, respectivamente.

Tabela 7. Tempo de execução (s) utilizando o *dataset* Yahoo-A1Benchmark

Detector	Rodada 1	Rodada 2	Rodada 3	Rodada 4	Rodada 5	Média
DASRS Rest	7	8	8	8	9	8,00
Windowed Gaussian	18	13	13	12	14	14,00
Relative Entropy	26	19	18	17	18	19,60
Bayesian Changepoint	25	22	20	20	21	21,60
CAD OSE	56	53	52	54	53	53,60
earthgecko Skyline	74	61	61	61	61	63,60
KNN CAD	79	69	65	70	65	69,60
EXPoSE	81	66	70	62	63	68,40
Numenta	212	188	181	192	178	190,20
Etsy Skyline	787	639	625	684	652	677,40

Tabela 8. Tempo de execução (s) utilizando o *dataset* NAB

Detector	Rodada 1	Rodada 2	Rodada 3	Rodada 4	Rodada 5	Média
DASRS Rest	23	24	25	22	24	23,60
Windowed Gaussian	45	49	49	45	47	47,00
Relative Entropy	62	65	66	63	63	63,80
Bayesian Changepoint	145	152	147	150	145	147,80
EXPoSE	222	230	228	224	218	224,40
CAD OSE	289	295	303	287	282	291,20
earthgecko Skyline	564	568	691	575	547	589,00
Numenta	696	768	738	791	688	736,20
KNN CAD	716	728	812	833	700	757,80
Etsy Skyline	16365	16072	16589	16497	16524	16409,40

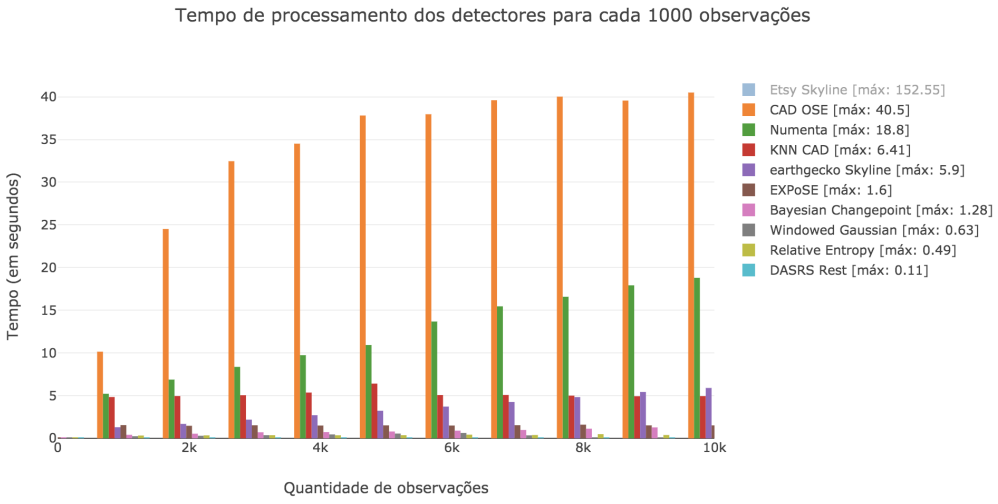


Figura 3. Tempo gasto pelos detectores no processamento de 1.000 observações

Os resultados da Figura 3 mostram que o DASRS Rest é 4 vezes mais rápido que o melhor resultado obtido pelos algoritmos do estado arte existentes no *framework* NAB (resultado do algoritmo Relative Entropy). Além disso, os algoritmos Numenta e earthgecko Skyline apresentaram tempo de processamento crescente a cada iteração, até o processamento da observação de número 10.000. O CAD OSE, por outro lado, apresentou tempo de processamento crescente apenas até o processamento da observação 8.000. Depois disso, seu tempo de processamento se manteve constante. A análise dos resultados da Figura 4 mostra que não há aumento no tempo de processamento à medida

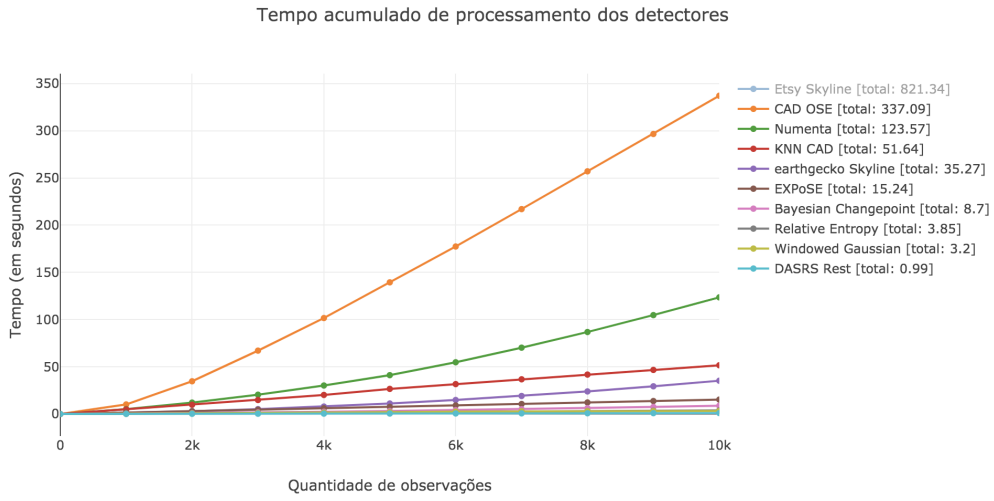


Figura 4. Tempo acumulado gasto pelos detectores no processamento de 10.000 observações

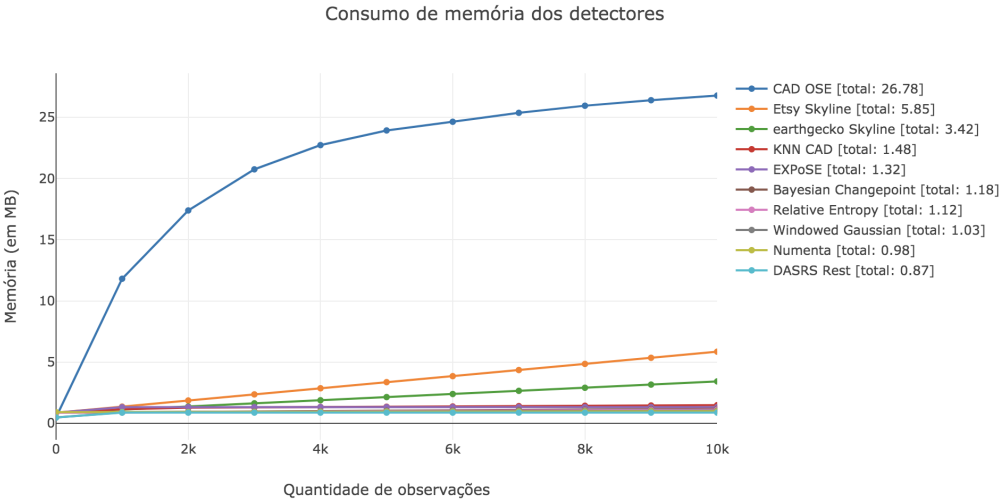


Figura 5. Memória utilizada pelos detectores no processamento de 10.000 observações

que mais dados são analisados pelo algoritmo DASRS Rest. Por outro lado, o tempo acumulado de processamento dos algoritmos CAD OSE e Numenta crescem em função do aumento na quantidade de observação. Essa característica pode inviabilizar a utilização desses três algoritmos no processamento de dados em *streaming* ou quando o algoritmo de detecção precisa processar simultaneamente um grande número de séries temporais. Retiramos a curva do algoritmo Etsy Skyline das figuras 3 e 4 porque obteve tempos muito acima dos demais.

Na quarta etapa avaliamos a quantidade de memória utilizada pelos detectores de anomalias. Utilizamos uma série temporal com 10.000 observações e o *benchmark asizeof*, que está detalhado em "ActiveState Code » Recipes"[5], para medir a memória

[5] Acessível em http://code.activestate.com/recipes/546530/

utilizada por cada um dos detectores a cada 1.000 observações analisadas. Os resultados da figura 5 mostram que os algoritmos CAD OSE, Etsy Skyline e earthgecko Skyline apresentaram aumento constante no consumo de memória que é, inclusive, maior que a dos demais algoritmos avaliados. O DASRS Rest, por outro lado, não apresenta variação no consumo de memória que é o menor (0,87 MB) entre todos os algoritmos analisados.

6. Conclusão e Trabalhos Futuros

Data centers são ambientes dinâmicos e diversificados, onde a atualização de software e hardware é frequente e as cargas de trabalho das aplicações variam em função da sazonalidade. Por essa razão, eles demandam sistemas de monitoração eficientes com capacidade para antecipar falhas a fim de evitar os prejuízos financeiros que são provocados pela interrupção dos serviços hospedados. Este artigo propõe o algoritmo DASRS Rest que detecta anomalias de forma eficiente, sem que seja necessário o conhecimento prévio sobre o serviço monitorado, permitindo a prevenção antecipada de falhas. A proposta foi desenvolvida e avaliada utilizando o *framework* NAB, o *dataset* nativo do NAB e o *dataset* Yahoo-A1Benchmark. Os resultados experimentais mostram que o DASRS Rest apresentou o menor consumo de memória e o menor tempo de processamento em todos os cenários de teste. Além disso, o algoritmo proposto obteve a melhor acurácia quando o *dataset* Yahoo-A1Benchmark foi processado e ficou em terceiro lugar quando o *dataset* nativo do NAB é utilizado. O DASRS Rest é capaz de processar series temporais geradas por diferentes tipos de dispositivos. Por isso, também pode ser utilizado para processar métricas de series temporais de sensores e dispositivos de IoT (*Internet of Things*).

Como trabalhos futuros, pretende-se implementar um classificador de anomalias. Assim, será possível determinar quando uma anomalia detectada deve gerar alarmes que precisam ser criteriosamente avaliados pelos administradores de sistemas, tais quais as anomalias vinculadas a ataques que exploram vulnerabilidades nas aplicações.

Referências

Adams, R. P. and MacKay, D. J. (2007). Bayesian online changepoint detection. Technical report, University of Cambridge.

Ahmad, S., Lavin, A., Purdy, S., and Agha, Z. (2017). Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134 – 147.

Ahmad, S. and Purdy, S. (2016). Real-Time Anomaly Detection for Streaming Analytics. *CoRR*, abs/1607.02480.

Brockwell, P. J. and Davis, R. A. (2016). Introduction to time series and forecasting. In *Springer Texts in Statistics*. Springer International Publishing.

Burnaev, E. and Ishimtsev, V. (2016). Conformalized density - and distance-based anomaly detection in time-series data. *arXiv:1608.04585*.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15:1–15:58.

Claps, G. G., Svensson, R. B., and Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology*, 57:21–31.

Couto, R. S., Campista, M. E. M., and Costa, L. H. M. (2012). A reliability analysis of datacenter topologies. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1890–1895.

Dias, R. (2019). Detecção de anomalias nas métricas das monitorações de máquinas de um data center. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio.

Earthgecko (2019). Earthgecko skyline. https://github.com/earthgecko/skyline. Acessado em abril de 2019.

Etsy, I. (2013). Skyline anomaly detection system. https://github.com/etsy/skyline. Acessado em abril de 2019.

Fox, A. J. (1972). Outliers in time series. *Journal of the Royal Statistical Society. Series B*, 34:350–363.

Gabel, M., Schuster, A., Bachrach, R., and Bjørner, N. (2012). Latent fault detection in large scale services. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12.

Guha, S., Mishra, N., Roy, G., and Schrijvers, O. (2016). Robust random cut forest based anomaly detection on streams. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2712–2721. JMLR.org.

Kejariwal, A. (2015). Twitter engineering: Introducing practical and robust anomaly detection in a time series. https://bit.ly/2YIPeSA. Acessado em junho de 2019.

Lavin, A. and Ahmad, S. (2015). Evaluating real-time anomaly detection algorithms - the numenta anomaly benchmark. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 38–44.

Munir, M., Siddiqui, S., Dengel, A., and Ahmed, S. (2018). Deepant: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access*, PP:1–1.

Rosner, B. (1983). Percentage points for a generalized esd many-outlier procedure. *Technometrics*, 25:165–172.

Safin, A. and Burnaev, E. (2017). Conformal kernel expected similarity for anomaly detection in time-series data. *Advances in Systems Science and Applications*, 17:22–33.

Schneider, M., Ertel, W., and Ramos, F. T. (2016). Expected similarity estimation for large-scale batch and streaming anomaly detection. *CoRR*, abs/1601.06602.

Singh, N. and Olinsky, C. (2017). Demystifying numenta anomaly benchmark. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1570–1577.

Smirnov, M. (2016). Contextual anomaly detector. https://github.com/smirmik/CAD. Acessado em abril de 2019.

Wang, C., Viswanathan, K., Choudur, L., Talwar, V., Satterfield, W., and Schwan, K. (2011). Statistical techniques for online anomaly detection in data centers. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 385–392.