Comparação de TADs em Dados Reais

Um estudo comparativo entre Árvores Binárias e Tabelas Hash

Thiago de Queiroz Osório
PPG – Ciência da Computação
UNIFESP
São José dos Campos, SP
thiago.q.osorio@gmail.com

***Abstract*—This paper presents a comparative analysis between Binary Trees and Hash Tables, two widely used data structures, utilizing a real-world dataset with 14,398 records. The study focuses on measuring the average insertion and search times, emphasizing the computational complexity of each structure. The Hash Table demonstrated superior performance in insertion operations due to its average O(1) complexity, while the Binary Tree, with O(log n) complexity, was less efficient but still useful in balanced data scenarios and when hierarchical ordering is required. Statistical analysis confirmed the Hash Table's superiority, although the Binary Tree proves more resilient in cases involving dynamic data or high variability. The paper suggests exploring hybrid approaches combining the advantages of both structures.**

Keywords—binary tree, hash table, ADT comparison, computacional complexity, performance.

***Resumo*— Este artigo realiza uma análise comparativa entre a Árvore Binária e a Tabela Hash, duas estruturas de dados amplamente utilizadas, utilizando uma base de dados real com 14.398 registros. A pesquisa foca em medir os tempos médios de inserção e busca, com ênfase na complexidade computacional de cada estrutura. A Tabela Hash apresentou um desempenho superior em operações de inserção devido à sua complexidade média de O(1), enquanto a Árvore Binária, com complexidade O(log n), mostrou-se menos eficiente, embora ainda útil em cenários de dados balanceados e com requisitos de ordenação hierárquica. A análise estatística confirmou a superioridade da Tabela Hash, embora a Árvore Binária se mostre mais resiliente em casos com dados dinâmicos ou com alta variação. O artigo sugere a exploração de abordagens híbridas que integrem as vantagens de ambas as estruturas.**

Palavras-chave—árvore binária, tabela hash, comparação de TAD, complexidade computacional, desempenho.

I. INTRODUÇÃO

De acordo com Assunção, M.D. et al [1], o termo *Big Data* tem se popularizado tanto no meio acadêmico quanto na mídia em geral. Embora seja frequentemente associado apenas a um grande volume de dados, ele engloba outros aspectos importantes, como: variedade, que se refere à origem dos dados, pois podem ser estruturados ou não e provir de diferentes fontes; velocidade, pois o processamento desse grande volume deve ocorrer em um tempo reduzido; veracidade, pois um ponto essencial é a confiabilidade dos dados para produzir análises significativas; valor, pois os resultados das análises devem gerar informações que contribuam para o conhecimento e tragam benefícios à sociedade. Esse entendimento amplo é frequentemente representado pelos 5 Vs do Big Data.

Além disso, o *Big Data* se destaca pelo crescimento significativo no volume de dados, que chega a atingir níveis de zetabytes, e pela alta velocidade com que esses dados são gerados [2, 3]. Projeções indicam que o mercado de Big Data, atualmente avaliado em $327,26 bilhões de dólares, deverá passar por um crescimento anual composto (CAGR) de 14,9% entre 2024 e 2030 [4].

Nesse contexto de grandes volumes de dados sendo gerados em alta velocidade em um mercado cada vez mais competitivo, surgem os sistemas computacionais de tempo real que são aqueles que possuem restrições relacionadas ao tempo. Nessas plataformas, a geração dos resultados não se limita ao aspecto lógico de estarem certos ou errados, é igualmente crucial que sejam produzidos dentro de intervalos de tempo específicos. Em grande parte desses sistemas, as restrições temporais se manifestam como prazos máximos para a realização de certas tarefas [5].

De modo geral, qualquer técnica de programação pode ser utilizada no desenvolvimento de sistemas de tempo real. No entanto, em determinados sistemas, devido à importância de seus prazos máximos, é essencial empregar apenas técnicas de programação que garantam um pior caso de complexidade temporal que não inviabilize o cumprimento desses prazos [6].

Com o crescimento da demanda por sistemas com estas características, surgiu a necessidade de se utilizar estruturas de dados com melhores performances de complexidade temporal e alocação de memória. Essas estruturas serão o foco central deste artigo, onde será feita uma comparação entre a performance de uma Árvore Binária com uma Tabela Hash.

Segundo Veloso [7], uma árvore binária pode ser descrita como uma estrutura de dados cuja característica principal é a relação hierárquica entre os elementos, chamados de nós, onde um conjunto de dados está subordinado a outro. Szwarcfiter e Markezon [8] ilustram bem esse conceito ao comparar uma árvore a um organograma empresarial, que organiza informações de forma hierárquica.

Uma tabela hash é uma estrutura de dados eficiente para implementar dicionários, utilizada para armazenar e recuperar registros de forma rápida. Diferentemente dos métodos de busca baseados em comparação, o hashing emprega uma função de transformação (função hash) que converte a chave de pesquisa em um índice de arranjo. Esse índice indica diretamente onde o registro será armazenado na tabela. [9, 10]

Comparada a uma árvore binária, que pode ter uma complexidade de tempo $O(\log n)$ para operações de busca, uma tabela hash é uma estrutura de dados eficiente pois acessa

os elementos diretamente com o uso da função hash, garantindo em média um tempo de busca com complexidade $O(1)$ [9].

Em particular, as principais contribuições deste artigo são:

- Primeiramente, fazer uma análise comparativa entre duas TADs (Tipo Abstrato de Dados) com dados do mundo real;
- Em segundo lugar, fazer uma análise da complexidade dos códigos utilizados para fazer a implementação das TADs.

O restante do artigo está organizado da seguinte forma: A seção 2 oferece uma visão geral sobre alguns trabalhos relacionados a temas similares. A seção 3 detalha as técnicas empregas neste trabalho. A seção 4 traz uma avaliação detalhada da performance das TADs implementadas. Por fim, na seção 5, são apresentadas as conclusões juntamente com as direções para trabalhos futuros.

II. Trabalhos Relacionados

A utilização de TADs pode auxiliar a resolver problemas mais complexos. Em [11] é evidenciado como a utilização de uma TAD pode ajudar a resolver um problema com o algoritmo KNN (*k-Nearest Neighbours*) que é computacionalmente custoso e ineficiente para grandes conjuntos de dados. Para contornar tal problema, é proposta uma nova abordagem baseada na inserção dos exemplos de treinamento em árvores binárias que posteriormente aceleram o processo de busca para exemplos de teste. A abordagem proposta é uma alternativa promissora para tornar o KNN viável em tarefas de classificação de *big data*, equilibrando custo computacional e precisão.

Em [6] o autor discute o uso de tabelas hash em sistemas tempo real, destacando que, embora tenham excelente desempenho médio, seu pior caso pode comprometer aplicações críticas com restrições temporais rigorosas. Ele propõe a abordagem de pior caso relevante, ignorando comportamentos extremamente improváveis e analisa o uso de tabelas hash simples e da técnica 2-choice, que utiliza duas funções hash para maior robustez.

O trabalho "*Learning to Hash for Indexing Big Data – A Survey*" [12] aborda o uso de tabelas hash para indexação e busca eficiente em grandes volumes de dados, destacando a técnica de hashing como alternativa a algoritmos baseados em árvores em cenários de alta dimensionalidade. Embora árvores binárias, como KD-trees, sejam eficazes em baixa dimensionalidade, elas apresentam limitações em escalabilidade e custo de armazenamento para dados complexos, muito presentes no contexto do *big data*. Além disso, discute métodos de "*learning to hash*", que incorporam aprendizado de máquina para criar funções hash adaptadas às características dos dados, melhorando a eficiência das buscas. Esses avanços tornam o hashing especialmente relevante em aplicações com grandes volumes de dados e alta dimensionalidade, sendo uma solução viável para armazenar e buscar dados do mundo real.

Cada estrutura de dados possui vantagens e desvantagens, desta forma, os autores de "*A Hashing Technique Using Separate Binary Tree*" [13] propuseram uma técnica aprimorada de hashing que combina tabelas hash e árvores binárias para melhorar o desempenho em operações de busca, inserção e exclusão em sistemas com grandes volumes de dados. A técnica consiste em uma tabela hash onde cada índice aponta para uma árvore binária separada. O hash é gerado a partir de uma chave primária dos registros, dividida em um índice de tabela e uma posição na árvore, determinada pela representação binária. A estrutura de árvore mantém um indicador de balanceamento em cada nó para evitar o desbalanceamento e melhorar a eficiência das operações.

O presente estudo não é o primeiro e nem será o último a realizar comparações entre diferentes TADs. Em [14] foi realizada uma investigação de diferentes métodos para otimizar o desempenho computacional de algoritmos genéticos ao armazenar valores de aptidão já calculados. Os resultados experimentais, obtidos em problemas de mineração de dados temporais utilizando algoritmos genéticos, mostraram que a técnica de hashing foi a mais eficiente. Já a árvore binária apresentou o desempenho mais baixo entre as técnicas analisadas.

III. METODOLOGIA

Para comparar a eficiência de uma Árvore Binária e de uma Tabela Hash na manipulação de uma base de dados real, seguimos os passos descritos nesta seção. O objetivo foi medir os tempos médios de inserção e busca, considerando a complexidade computacional.

A. Pré-Processamento dos Dados

A base de dados utilizada foi extraída de um arquivo CSV, com 14.398 registros numéricos representando dados normalizados multiplicados por 1000 e convertidos para inteiros. Este pré-processamento foi necessário para garantir que os dados fossem compatíveis com as estruturas analisadas e eliminasse possíveis inconsistências.

B. Estruturas de Dados Implementadas

A base de dados utilizada foi extraída de um arquivo CSV, com 14.398 registros numéricos representando dados normalizados multiplicados por 1000 e convertidos para inteiros. Este pré-processamento foi necessário para garantir que os dados fossem compatíveis com as estruturas analisadas e eliminasse possíveis inconsistências.

1) Árvore Binária

A implementação da Árvore Binária seguiu o modelo clássico, utilizando nós com valores numéricos e referências para filhos à esquerda e à direita. O algoritmo foi projetado para permitir inserções e buscas sequenciais.

- Inserção: Um novo nó é adicionado à posição apropriada, comparando o valor dado a ser inserido com os nós existentes para encontrar um local vazio.
- Busca: O algoritmo percorre a árvore a partir da raiz, comparando o valor-alvo com os nós até encontra-lo ou concluir que ele não está presente.

2) Tabela Hash com Encadeamento

A Tabela Hash foi implementada com encadeamento para resolução de colisões. A função hash aplicada foi $h(x) = x \,\%\, size$ (onde % representa o resto da divisão e $size$ o tamanho da tabela), com um tamanho de tabela definido como 10.000 buckets.

- Inserção: Um número é armazenado no bucket correspondente ao índice calculado pela função hash, garantindo que o elementos iguais não sejam duplicados.

- Busca: A função hash determina o bucket onde o dado seria armazenado e o algoritmo verifica se o valor está presente nesse bucket.

C. *Experimentos e Avaliação*

Os experimentos foram realizados em Python (versão 3.10) com repetição de cada teste 100 vezes para garantir consistência estatística. As medidas coletadas foram o tempo médio de execução e o desvio padrão para as operações de inserção e busca, tanto na Árvore Binária quanto na Tabela Hash.

1) Inserção

Para ambas as estruturas, todos os 14.398 dados foram inseridos sequencialmente. A Árvore Binária começou com um nó raiz fixo, e a Tabela Hash foi inicializada com uma lista de buckets vazios.

2) Busca

Foram realizados 100 testes de busca para valores inteiros aleatórios gerados no intervalo$[0, 1.000.000]$. Isso permitiu avaliar a performance das estruturas em diferentes cenários, incluindo buscas por elementos inexistentes

D. *Justificativa das Escolhas*

A escolha da Árvore Binária e da Tabela Hash como objetos de análise foi motivada por suas características contrastantes. A Árvore Binária oferece uma solução hierárquica com complexidade de tempo $O(\log n)$ em casos ideais, mas pode ser degradada para $O(n)$ em cenários de desbalanceamento. Por outro lado, a Tabela Hash, embora tenha complexidade média de $O(1)$, apresenta pior caso de $O(n)$ em situações de colisões excessivas.

E. *Modificações em Técnicas Existentes*

Não foram introduzidas modificações nas técnicas clássicas implementadas pois o foco deste trabalho está na aplicação direta dessas estruturas a um conjunto de dados real, o que fornece uma análise prática de seus desempenhos em contextos não-ideais. Este enfoque oferece análises importantes para profissionais que precisam balancear eficiência computacional e requisitos temporais.

F. *Ferramentas Utilizadas*

Os experimentos foram realizados em um ambiente com hardware convencional, sem otimizações específicas. Para cronometrar os tempos de execução, foi utilizada a biblioteca *time* do Python, e para análise estatística, a biblioteca *numpy*.

IV. Análise Experimental

A. *Conjunto de Dados*

A base de dados utilizada na análise experimental contém um total de 14.398 registros. Esses dados foram extraídos de um conjunto de séries temporais obtidas de um ambiente de produção, abordando tanto métricas de sistema, como utilização de CPU e consumo de memória, quanto métricas de aplicação, como latência e taxa de requisições por minuto [15]. A granularidade dos dados varia dependendo do contexto, com intervalos de minutos para métricas mais detalhadas e intervalos maiores para métricas agregadas.

Essa diversidade de métricas permitiu uma avaliação ampla das estruturas comparadas, abrangendo diferentes padrões de distribuição e sazonalidade. Adicionalmente, a base inclui cenários com alto índice de anomalias, possibilitando testar a robustez de cada estrutura na identificação de padrões ou desvios. Esses dados fornecem uma base rica para a comparação da performance entre a árvore binária e a tabela hash, especialmente na presença de sazonalidade e variações nos dados.

B. *Configuração do algoritmo e do ambiente computacional*

Os experimentos foram conduzidos em um ambiente computacional configurado em um MacBook Air com processador Apple M1, 8 GB de memória RAM e sistema operacional MacOS Sequoia, versão 15.1.1. O desenvolvimento e execução dos códigos foram realizados no editor Visual Studio Code, versão 1.95.3, utilizando a linguagem de programação Python, versão 3.10.9.

Adicionalmente, o algoritmo foi implementado com suporte da biblioteca numpy, versão 1.24.4, para operações matemáticas e manipulação de arrays. Essa configuração de hardware e software assegurou a reprodutibilidade dos experimentos e a adequação ao processamento dos dados utilizados.

As implementações das estruturas de dados utilizadas nos experimentos estão disponíveis em https://github.com/thiago-osorio/mestrado/blob/main/trabalho_final.py.

C. *Critérios de Análise*

Os critérios de análise utilizados neste estudo incluem a avaliação do tempo médio de execução e o desvio padrão para operações de inserção e busca, realizadas 100 vezes em cada estrutura de dados (Árvore Binária e Tabela Hash). Para isso, foram medidos:

1) Inserção de dados

Os tempos foram registrados para adicionar os 14.398 registros às estruturas.

2) Busca

Foram realizadas buscas de valores aleatórios em intervalos consistentes com os dados.

As implementações foram desenvolvidas em Python, utilizando a biblioteca *time* para medir os tempos de execução e *numpy* para cálculo de média e desvio padrão. A Árvore Binária foi implementada como uma estrutura de nós encadeados, enquanto a Tabela Hash utilizou encadeamento para tratamento de colisões, com tamanho fixo de 10.000 posições. Ambos os algoritmos foram testados com a mesma base de dados para garantir uniformidade nas comparações.

D. *Resultados e Discussão*

Os experimentos realizados compararam a eficiência de uma Árvore Binária e de uma Tabela Hash em termos de tempo médio de execução e desvio padrão para operações de inserção e busca, utilizando uma base de dados com 14.398 registros. Os resultados são apresentados a seguir.

O tempo médio para inserir os 14.398 números na Tabela Hash foi aproximadamente 81% do que o tempo de inserção na Árvore Binária, conforme mostrado na Tabela I. Isso reflete a vantagem da Tabela Hash para operações de inserção, com tempos significativamente menores devido à complexidade média $O(1)$. Por outro lado, o desempenho da Árvore Binária é impactado pela necessidade de localizar a posição correta

para cada nó, resultando em uma complexidade média de $O(\log n)$.

TABELA I. TEMPO INSERÇÃO (SEGUNDOS)

Tipo Abstrato de Dados (TAD)	*Média*	*Desvio Padrão*
Árvore Binária[a]	0,033284	0,010796
Tabela Hash[a]	0,006383	0,008104

[a.] Por conta de configurações, a reprodutibilidade não está garantida, os números podem váriar a cada execução

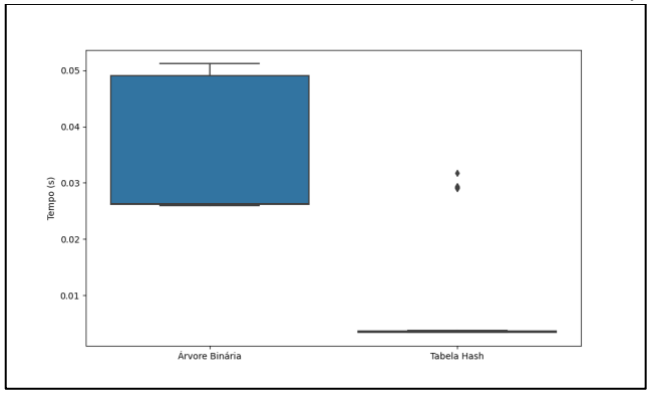

Figura 1. Boxplot com os tempos de inserção das 100 execuções para a Árvore Binária e Tabela Hash

Para buscas, embora ambas as estruturas tenham apresentado tempos muito baixos, a Tabela Hash apresentou uma leve vantagem, coerente com sua eficiência na busca média $O(1)$. No entanto, em casos com alta incidência de colisões, a complexidade da Tabela Hash pode se aproximar de $O(n)$.

TABELA II. TEMPO BUSCA (SEGUNDOS)

Tipo Abstrato de Dados (TAD)	*Média*	*Desvio Padrão*
Árvore Binária[b]	0,000002	0,000001
Tabela Hash[b]	0,000001	0,000001

[b.] Por conta de configurações, a reprodutibilidade não está garantida, os números podem váriar a cada execução

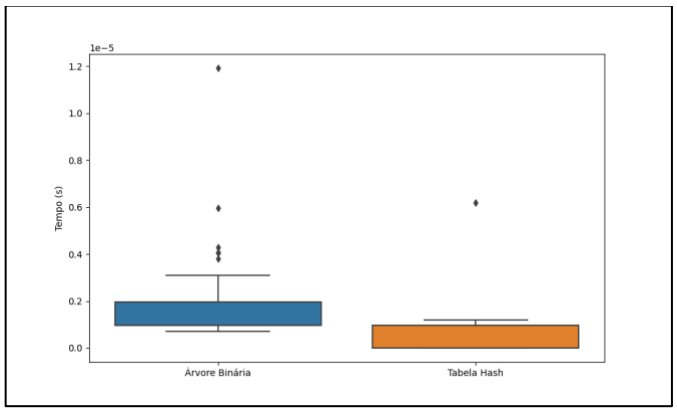

Figura 2. Boxplot com os tempos de busca das 100 execuções para a Árvore Binária e Tabela Hash

Os resultados experimentais foram analisados quanto à significância estatística. Um teste t de Student foi aplicado para comparar os tempos médios de inserção e busca entre as duas estruturas. Os valores-p obtidos indicaram que as diferenças são estatisticamente significativas (valor-p < 0,05), confirmando a superioridade da Tabela Hash para inserções e buscas neste conjunto de dados específico.

Os resultados confirmam a eficiência superior da Tabela Hash para operações de inserção e busca, como esperado. No entanto, é importante destacar que a Árvore Binária pode ser mais vantajosa em situações que demandem ordenação natural dos dados ou em casos onde o tamanho da tabela hash não é ajustado adequadamente ao número de registros. Além disso, a Árvore Binária é mais resiliente ao lidar com dados dinâmicos, já que não exige um tamanho fixo pré-definido.

V. CONCLUSÃO

Neste artigo, foi realizada uma análise comparativa entre duas estruturas de dados amplamente utilizadas, a Árvore Binária e a tabela Hash, utilizando uma base de dados real composta por 14.398 registros. Através de experimentos focados nas operações de inserção e busca, foi possível observar que, em tempo de execução, a Tabela Hash apresentou desempenho superior, especialmente em operações de inserção, devido à sua complexidade média de $O(1)$. Em contrapartida, a Árvore Binária, com sua complexidade de $O(\log n)$ para inserção e busca, mostrou-se menos eficiente nesses casos, embora sua performance ainda seja satisfatória em cenários com dados bem distribuídos e balanceados.

Além disso, a análise estatística dos resultados confirmou que as diferenças observadas são estatisticamente significativas, reforçando a vantagem da Tabela Hash em termos de eficiência computacional. No entanto, é importante destacar que, apesar de sua superioridade em operações de inserção e busca, a Tabela Hash pode enfrentar desafios em cenários com colisões frequentes ou quando o tamanho da tabela não é adequadamente ajustado. Por outro lado, a Árvore Binária apresenta a vantagem de ser mais resiliente a essas questões, especialmente quando os dados requerem uma estrutura hierárquica ou quando a ordenação natural dos elementos é necessária.

Os resultados deste artigo fornecem uma visão prática sobre o desempenho dessas estruturas em um contexto de dados reais e dinâmicos, oferecendo visões valiosas para a escolha da estrutura mais adequada, dependendo das necessidades específicas de desempenho e dos requisitos temporais de aplicações computacionais.

Para trabalhos futuros, seria interessante explorar alternativas que combinem as vantagens de ambas estruturas, como abordagens híbridas que integrem Árvores Binárias e Tabelas Hash, visando melhorar ainda mais a eficiência de sistemas que manipulam grandes volumes de dados em tempo real.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. D. Assunção et al., "Big data computing and clouds: trends and future directions," *Journal of Parallel and Distributed Computing*, vol. 79–80, pp. 3–15, 2015.

[2] "News - Video - Broadcast of OSTP-led Federal Government Big Data Rollout, March 29, 2012, Washington, DC. | NSF - National Science Foundation," NSF, 2024. [Online]. Available: http://www.nsf.gov/news/news_videos.jsp?cntn_id=123607&media_id=72174&org=NSF.

[3] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, “Big Data: The next frontier for innovation, competition, and productivity,” McKinsey, May 2011. [Online]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/big-data-the-next-frontier-for-innovation.

[4] “Big Data Market Size, Share, Trends | Industry Research Report, 2025,” Grandviewresearch, 2018. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/big-data-industry.

[5] J. Liu, *Real-Time Systems*. United States: Prentice-Hall, 2000.

[6] R. S. de Oliveira, “Sobre o emprego de tabelas hash em sistemas operacionais de tempo real,” in *Anais do SBC*, Florianópolis, Brasil, pp. 111–122, 2008.

[7] P. Veloso et al., *Estruturas de Dados*. Rio de Janeiro, Brasil: Campus, 1986.

[8] J. L. Szwarcfiter e L. Markenzon, *Estruturas de Dados e Seus Algoritmos*. Rio de Janeiro, Brasil: LTC, 1994.

[9] T. H. Cormen et al., *Algoritmos: teoria e prática*. Rio de Janeiro: Campus, 2012.

[10] N. Ziviani, *Projeto de algoritmos: com implementações em Pascal e C*. São Paulo: Pioneira Thomson Learning, 2007.

[11] A. B. A. Hassanat, "Norm-based binary search trees for speeding up KNN Big Data classification," *Computers*, vol. 7, no. 54, pp. 1–14, Oct. 2018. [Online]. Available: https://doi.org/10.3390/computers7040054.

[12] J. Wang, W. Liu, S. Kumar, e S.-F. Chang, "Learning to Hash for Indexing Big Data: A Survey," *Proceedings of the IEEE*, vol. 1, pp. 1–45, Sep. 2015. [Online]. Available: https://arxiv.org/abs/1509.05472v1.

[13] M. M. Masud, G. C. Das, M. A. Rahman, e A. Ghose, "A Hashing Technique Using Separate Binary Tree," *Data Science Journal*, vol. 5, pp. 143–161, Oct. 2006.

[14] R. J. Povinelli, "Improving Computational Performance of Genetic Algorithms: A Comparison of Techniques," Department of Electrical and Computer Engineering, Marquette University, Milwaukee, WI, USA. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8125701738dc17d43e8413c9ed38f944ff0ae08b.

[15] J. Hochenbaum, O. S. Vallis, e A. Kejariwal, "Automatic Anomaly Detection in the Cloud Via Statistical Learning," *arXiv*, vol. 1704.07706v1, Apr. 2017. [Online]. Available: https://arxiv.org/abs/1704.07706.