Otimização Multiobjetivo de Configurações de Produtores do Apache Kafka para Redução de Latência e Aumento de Vazão

Paulo Cesar dos Santos da Silva, Ian Vilar Bastos, Marcelo Gonçalves Rubinstein

[1]PEL/DETEL-FEN
Universidade do Estado do Rio de Janeiro

cesar.ensino@matematicaeletronica.com.br, {ian, rubi}@uerj.br

Abstract. *Apache Kafka is one of the most widely used data streaming solutions nowadays. This solution is based on the publish-subscribe model, in which producers publish data and consumers receive it. Configuring producers involves multiple interdependent parameters that influence metrics such as throughput, latency, and latency variability, characterizing a multi-objective optimization problem. This work proposes an approach that integrates surrogate predictive modeling using XGBoost (eXtreme Gradient Boosting) with multi-objective optimization based on the NSGA-III (Non-dominated Sorting Genetic Algorithm III) evolutionary algorithm. The problem considers four objectives: minimizing latency, maximizing throughput, and minimizing the variability of both latency and throughput in the communication between a producer and a Kafka broker. The solutions are evaluated through experiments conducted with five message sizes. The results indicate a fast convergence of NSGA-III and a strong agreement between estimated and real values, especially for throughput and latency.Furthermore, the gains obtained are more than 10% in terms of throughput and 40% in terms of latency when comparing the performance of a recommended configuration resulting from optimization and of the default Kafka configuration.*

Resumo. *O Apache Kafka é uma das soluções de streaming de dados mais amplamente utilizadas atualmente. Essa solução é baseada no modelo publicação/assinatura, no qual produtores publicam dados e consumidores os recebem. A configuração dos produtores envolve múltiplos parâmetros interdependentes que influenciam métricas como vazão, latência e variabilidade da latência, caracterizando um problema de otimização multiobjetivo. Este trabalho propõe uma abordagem que integra modelagem substituta preditiva com XGBoost (eXtreme Gradient Boosting) e otimização multiobjetivo baseada no algoritmo evolutivo NSGA-III (Non-dominated Sorting Genetic Algorithm III). O problema considera quatro objetivos: minimizar a latência, maximizar a vazão e minimizar as variabilidades da latência e da vazão na comunicação entre um produtor e um cluster do Kafka. As soluções são avaliadas por meio de experimentos conduzidos com cinco tamanhos de mensagem. Os resultados indicam uma rápida convergência do NSGA-III e uma aderência entre valores estimados e reais, especialmente para a vazão e para a latência. Além disso, os ganhos obtidos são de mais de 10% em termos de vazão e 40% em termos de latência quando são comparados os desempenhos entre uma configuração proveniente da otimização e a configuração padrão do Kafka.*

1. Introdução

O processamento de fluxo de dados tornou-se um componente fundamental em arquiteturas modernas orientadas a eventos, sendo amplamente utilizado em aplicações como monitoramento em tempo real, IoT (*Internet of Things*), sistemas financeiros e análise contínua de dados [Fu and Soman 2021]. A evolução desses sistemas ao longo das últimas décadas resultou em plataformas altamente escaláveis e distribuídas, capazes de operar com altas taxas de dados e com requisitos rigorosos de latência e disponibilidade [Kreps et al. 2011].

Nesse contexto, o Apache Kafka consolidou-se como uma das principais soluções para *data streaming*, oferecendo um modelo de comunicação publicação/assinatura escalável e tolerante a falhas, amplamente adotado em ambientes industriais e acadêmicos [Raptis and Passarella 2023]. Sua capacidade de processar grandes volumes de dados em tempo real o torna essencial em cenários que exigem rápidas tomadas de decisões baseadas em dados contínuos.

Entretanto, o desempenho do Kafka é altamente dependente da configuração de seus parâmetros. Parâmetros como *batch.size*, *linger.ms*, *acks* e *compression.type* impactam diretamente métricas críticas como vazão e latência. A literatura aponta que a escolha inadequada de valores para esses parâmetros pode comprometer o desempenho do sistema, especialmente em cenários com cargas variáveis e padrões de tráfego não determinísticos [Bao et al. 2018].

Apesar da ampla adoção do Kafka [Raptis and Passarella 2023], ainda existe uma lacuna na exploração sistemática do espaço de configuração desses parâmetros, considerando múltiplos objetivos. Trabalhos relacionados concentram-se majoritariamente em desempenho médio (vazão e latência), conforme observado em abordagens de otimização e ajuste automático de sistemas distribuídos [Ma et al. 2025, Dou et al. 2024], negligenciando as variabilidades das métricas, que são fundamentais para garantir previsibilidade em sistemas sujeitos a SLAs (*Service Level Agreements*) rigorosos. Além disso, abordagens tradicionais de avaliação exigem um grande volume de experimentação real [Bao et al. 2018], tornando o processo custoso e pouco escalável.

Diante desse cenário, este trabalho investiga se a combinação de um modelo substituto preditivo com uma otimização multiobjetivo é capaz de identificar configurações eficientes dos produtores do Kafka, reduzindo o custo experimental e incorporando métricas de desempenho e de estabilidade[1] de forma conjunta. Para isso, propõe-se uma abordagem que integra uma estratégia de planejamento experimental baseada em *Design of Experiments* (DOE), utilizando o método *Latin Hypercube Sampling* (LHS) para geração de amostras representativas do espaço de parâmetros [Montgomery 2017], uma modelagem preditiva com XGBoost e uma otimização multiobjetivo com NSGA-III. O modelo considera quatro objetivos: minimizar o percentil-95 (p95) da latência, minimizar a variabilidade da latência, maximizar a vazão e minimizar a variabilidade da vazão. As principais contribuições deste trabalho são:

- uma formulação do problema de configuração dos produtores do Kafka como um problema de otimização multiobjetivo;

[1] A estabilidade neste artigo se refere à variação das métricas de desempenho. Visa-se uma pequena variação das métricas.

- o uso de um modelo substituto baseado em XGBoost para estimar o desempenho e reduzir o custo de avaliação das configurações;
- a incorporação explícita de métricas de variabilidade como critérios de otimização, considerando desempenho e estabilidade de forma conjunta;
- uma avaliação experimental em ambiente real, considerando múltiplos tamanhos de mensagem.

Os resultados da avaliação de desempenho demonstram que a abordagem proposta permite identificar configurações eficientes, evidenciando os compromissos entre vazão, latência e estabilidade, além de reduzir significativamente o número de experimentos necessários. A melhoria obtida para uma configuração proveniente da otimização resulta em ganhos de mais de 10% e 40% na vazão e na latência, respectivamente, em comparação com uma configuração padrão do Kafka.

O restante deste artigo está organizado da seguinte forma. A Seção 2 descreve os trabalhos relacionados. A Seção 3 apresenta uma visão geral do Apache Kafka e dos principais parâmetros considerados neste estudo. A Seção 4 apresenta o modelo substituto baseado em XGBoost enquanto a Seção 5 introduz o algoritmo NSGA-III. A Seção 6 apresenta a solução proposta. A Seção 7 descreve o ambiente experimental e a metodologia adotada. A Seção 8 apresenta e discute os resultados obtidos. Por fim, a Seção 9 apresenta as conclusões e direções para trabalhos futuros.

2. Trabalhos Relacionados

A otimização de sistemas de processamento de fluxo de dados tem sido amplamente investigada, sobretudo devido à alta dimensionalidade do espaço de configuração e à forte influência dos parâmetros no desempenho. No contexto do Apache Kafka, estudos indicam que o desempenho do sistema depende diretamente da adequada configuração de parâmetros como tamanho de lote de mensagens, latência de envio e mecanismos de confirmação, cuja interação é não linear e de difícil modelagem [Raptis and Passarella 2023, Apache Software Foundation 2026]. Essa complexidade torna inviável a exploração exaustiva do espaço de busca em cenários reais.

Nesse contexto, abordagens baseadas em metaheurísticas têm sido propostas para automatizar a busca por configurações eficientes. O trabalho de Ma et al. [Ma et al. 2025] propõe o algoritmo DMGA-PSO, que combina algoritmos genéticos e otimização por enxame de partículas para maximizar o desempenho em sistemas de mensageria distribuída, mais precisamente no Kafka. Esses métodos são eficazes na exploração do espaço de soluções; contudo, ainda dependem de múltiplas avaliações reais, o que pode ser custoso.

Para mitigar esse problema, modelos preditivos têm sido empregados como substitutos do sistema real, permitindo estimar métricas de desempenho sem a necessidade de experimentação extensiva. Abordagens desse tipo são amplamente discutidas na literatura especializada [Jin 2011] e têm sido aplicadas em cenários de ajuste automático de parâmetros, como no trabalho de Bao et al. [Bao et al. 2018], que utiliza aprendizado de máquina para reduzir o número de experimentos necessários.

Em ambientes distribuídos mais amplos, como plataformas de processamento em nuvem e *streaming*, observa-se a integração entre otimização multiobjetivo e modelos preditivos. Métodos como MOEA/D [Zhang and Li 2007] e algoritmos evolutivos mais

recentes, como o NSGA-III [Deb and Jain 2013], permitem lidar com múltiplos objetivos conflitantes. Frameworks como o COTuner [Dou et al. 2024] utilizam otimização bayesiana para ajustar simultaneamente parâmetros de software e recursos computacionais, reduzindo o custo experimental no Kafka.

Apesar desses avanços, a maioria dos trabalhos concentra-se em métricas médias de desempenho, como vazão e latência, sem considerar explicitamente a variabilidade dessas métricas. No entanto, em aplicações reais, especialmente sob restrições de SLA, a estabilidade do sistema é tão importante quanto seu desempenho médio. Além disso, ainda são limitadas as abordagens que integram, de forma conjunta, modelos substitutos e otimização multiobjetivo em cenários reais de sistemas de *streaming*, como proposta por este trabalho e que considera simultaneamente latência, vazão e suas variabilidades.

3. O Sistema Distribuído Apache Kafka

O Apache Kafka é uma plataforma distribuída de *streaming* projetada para alta escalabilidade, tolerância a falhas e baixa latência [Kreps et al. 2011]. Sua arquitetura segue o modelo publicação/assinatura, no qual produtores enviam registros a *brokers* e consumidores realizam a leitura dos dados nesses *brokers*.

Os dados são organizados em tópicos, que representam agrupamentos lógicos de mensagens de acordo com uma categoria ou fluxo de dados. Cada tópico é subdividido em partições, que constituem a unidade básica de paralelismo e armazenamento no Kafka. Uma partição pode ser entendida como uma sequência ordenada e imutável de registros, onde cada mensagem recebe um identificador sequencial denominado *offset*. Essa estrutura permite que diferentes partições de um mesmo tópico sejam distribuídas entre distintos *brokers* do cluster, viabilizando o balanceamento de carga e o processamento paralelo [Raptis and Passarella 2023]. Além disso, a replicação das partições entre *brokers* garante tolerância a falhas e alta disponibilidade.

O desempenho do sistema depende tanto da infraestrutura quanto das configurações adotadas. Este trabalho foca na otimização dos parâmetros dos produtores, que influenciam diretamente a vazão e a latência [Kang et al. 2023]. Os parâmetros considerados são:

- ***batch.size***: tamanho máximo do lote de mensagens. As mensagens são enviadas em lotes. Valores maiores tendem a aumentar a vazão, ao custo de maior latência;
- ***linger.ms***: tempo máximo de espera para a formação de lotes. Um lote pode ser enviado antes que seu tamanho máximo seja alcançado; basta que esse temporizador estoure. Valores maiores favorecem a vazão, enquanto valores menores reduzem a latência;
- ***acks***: nível de confirmação do envio. Maior confiabilidade implica maior latência e menor vazão;
- ***compression.type***: algoritmo de compressão aplicado aos dados. Uma maior compressão reduz o uso da rede, mas aumenta o consumo de CPU.

Esses parâmetros apresentam interdependências complexas e efeitos não lineares sobre as métricas de desempenho. Ajustes que favorecem a vazão podem impactar negativamente a latência e vice-versa, além de influenciar a estabilidade temporal do sistema. Esse comportamento caracteriza um espaço de busca multidimensional e altamente correlacionado, tornando inviável a obtenção de configurações ótimas por ajuste manual.

Dessa forma, a configuração de um produtor Kafka pode ser naturalmente formulada como um problema de otimização multiobjetivo, no qual diferentes métricas conflitantes devem ser consideradas simultaneamente, justificando o uso de técnicas automatizadas de busca no espaço de configurações. Além disso, o uso de um modelo substituto permite estimar o desempenho e reduzir o custo de avaliação das configurações.

4. O Modelo Substituto Baseado em XGBoost

No contexto de sistemas distribuídos, como o Apache Kafka, o relacionamento entre parâmetros de configuração e métricas de desempenho é altamente não linear e difícil de modelar analiticamente, sendo frequentemente tratado como um problema de "caixa preta". A avaliação direta de cada configuração exige a execução do sistema real, o que implica alto custo computacional e tempo de experimentação.

Para contornar essa limitação, este trabalho utiliza o XGBoost como modelo substituto (*surrogate model*) para estimar métricas como vazão, latência e suas variabilidades a partir dos parâmetros de configuração dos produtores.

O XGBoost é um algoritmo de aprendizado supervisionado baseado em *gradient boosting* que combina múltiplas árvores de decisão para realizar predições com alta precisão [Chen and Guestrin 2016]. O modelo constrói, iterativamente, um conjunto de árvores, no qual cada nova árvore busca corrigir os erros das anteriores, minimizando uma função de perda regularizada.

A Equação 1 apresenta a predição final do modelo:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \tag{1}$$

onde x_i representa o vetor de características da i-ésima amostra, correspondente a uma configuração do produtor Kafka e ao tamanho da mensagem considerado. f_k representa uma árvore de decisão e K corresponde ao número total de árvores no modelo.

A escolha do XGBoost deve-se à sua capacidade de modelar relações não lineares, lidar com interações complexas entre variáveis e apresentar um bom desempenho, mesmo com conjuntos de dados relativamente pequenos, características que são adequadas ao problema abordado.

Integrado ao processo de otimização, o modelo substituto permite que o algoritmo evolutivo utilizado neste trabalho avalie rapidamente o desempenho de um grande número de soluções candidatas sem a necessidade de utilização do sistema real a cada iteração, reduzindo significativamente o custo computacional do processo e tornando viável a exploração do espaço de configurações.

5. O Algoritmo Evolutivo Multiobjetivo NSGA-III

Algoritmos evolutivos são métodos de otimização inspirados na evolução natural, nos quais soluções candidatas evoluem ao longo de gerações por meio de operadores como seleção, cruzamento e mutação.

Em problemas com múltiplos objetivos conflitantes, como no caso da otimização proposta, uma comparação direta entre as soluções não é trivial. Logo busca-se uma outra

forma de comparação. Uma técnica bastante utilizada, denominada dominância de Pareto, corresponde a achar um conjunto de soluções não dominadas (fronteira de Pareto). Uma solução A domina B se é melhor ou igual em todos os objetivos e estritamente melhor em pelo menos um, como demonstrado na Equação 2:

$$f_i(x_a) \leq f_i(x_b)\ \forall i \quad \text{e} \quad f_i(x_a) < f_i(x_b) \text{ para algum } i. \tag{2}$$

O NSGA-III é um algoritmo evolutivo projetado para problemas com múltiplos objetivos, especialmente quando o número de objetivos é superior a três, nos quais métodos tradicionais como o NSGA-II apresentam dificuldades em manter a diversidade na população [Deb and Jain 2013]. Para isso, o algoritmo utiliza pontos de referência distribuídos no espaço de objetivos, que orientam a seleção das soluções ao longo da fronteira de Pareto, garantindo diversidade e a distribuição uniforme das soluções. Com esses pontos, por exemplo, é possível escolher, na fronteira de Pareto, o maior valor para um determinado objetivo ou valores equilibrados entre eles.

De forma geral, o NSGA-III começa com uma população de soluções candidatas, que são avaliadas de acordo com os objetivos definidos. A cada geração, operadores evolutivos são aplicados para gerar novos indivíduos, e a seleção é realizada com base na dominância de Pareto e na associação aos pontos de referência.

No contexto deste trabalho, a configuração do produtor Kafka é formulada como um problema multiobjetivo com quatro objetivos conflitantes: minimizar a latência p95, minimizar a variabilidade da latência, maximizar a vazão e minimizar a variabilidade da vazão. A presença simultânea desses objetivos torna o problema particularmente complexo, justificando o uso do NSGA-III para explorar de forma eficiente o espaço de soluções e identificar configurações que representem diferentes compromissos entre desempenho e estabilidade.

6. A Solução Proposta

A Figura 1 ilustra o funcionamento da solução proposta. Inicialmente, configurações reais são utilizadas como entrada do método *Design of Experiments*, e como saída é obtido um conjunto reduzido de configurações. Em seguida, esse conjunto é utilizado para treinar um modelo substituto baseado em XGBoost, responsável por estimar as métricas de desempenho do sistema.

O modelo treinado é, então, integrado ao algoritmo NSGA-III, que inicializa uma população de soluções candidatas. A cada geração, novos indivíduos são produzidos por operadores evolutivos e avaliados por meio do modelo substituto, que prediz métricas como latência, vazão e variabilidade de ambas.

Com base nessas predições, as soluções são ordenadas segundo a dominância de Pareto e associadas a pontos de referência, garantindo diversidade na população. É realizada, então, a seleção da próxima geração. Esse processo iterativo continua até que o critério de parada seja atingido.

Ao final, é obtida uma fronteira de Pareto contendo configurações não dominadas. A partir dessa fronteira, soluções representativas são selecionadas e validadas no sistema real, permitindo verificar a aderência entre os valores estimados e observados, o que é

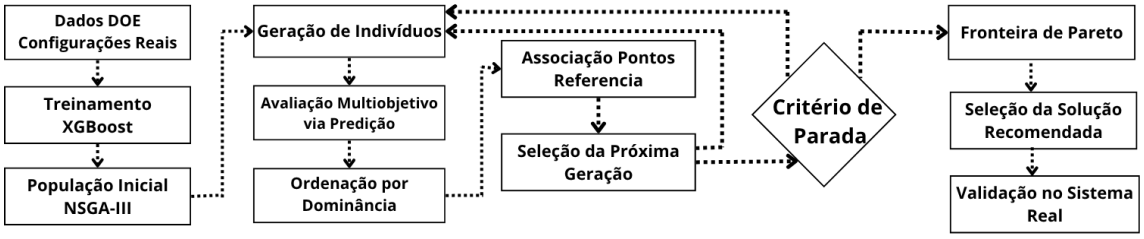

Figura 1. Fluxograma da solução.

fundamental para validar a capacidade preditiva do modelo e garantir que as soluções obtidas sejam efetivamente aplicáveis em cenários reais.

6.1. Formulação do Problema

O problema de otimização da configuração do produtor Kafka é modelado como um problema de otimização multiobjetivo, no qual o objetivo é encontrar combinações de parâmetros que maximizem o desempenho e a estabilidade do sistema sob diferentes condições de carga.

A Equação 3 define o vetor de decisão:

$$x = (batch.size, linger.ms, acks, compression), \tag{3}$$

cujos domínios são definidos como: $batch.size \in [1024, 262144]$, $linger.ms \in [0, 100]$, $acks \in \{0, 1, all\}$ e $compression \in \{none, zstd, gzip, snappy, lz4\}$. Os limites seguem a documentação oficial do Kafka [Apache Software Foundation 2026], abrangendo desde valores mínimos até limites que viabilizam a exploração dos compromissos entre vazão, latência e estabilidade. O problema considera diferentes tamanhos de mensagens. A Equação 4 representa o tamanho da mensagem:

$$w = (record\_size). \tag{4}$$

O conjunto de expressões 5 define os objetivos:

$$\begin{aligned} &\min f_1(x) = \text{Lat}_{p95}(x), \\ &\min f_2(x) = -\text{Vazão}(x), \\ &\min f_3(x) = CV(\text{Lat}(x)), \\ &\min f_4(x) = CV(\text{Vazão}(x)), \end{aligned} \tag{5}$$

onde: f_1 representa a latência p95, f_2 corresponde à maximização da vazão, f_3 representa a variabilidade definida como o Coeficiente de Variação (*Coefficient of Variation* - CV) da latência, e f_4 representa a variabilidade da vazão.

O problema está sujeito à seguinte restrição, definida na Inequação 6:

$$batch.size \geq record\_size, \tag{6}$$

garantindo que o tamanho do lote seja suficiente para acomodar as mensagens geradas.

Essa formulação caracteriza um problema multiobjetivo com objetivos conflitantes, no qual melhorias na vazão podem implicar aumento da latência ou da variabilidade, tornando necessária a busca por soluções de compromisso na fronteira de Pareto.

7. Metodologia

Esta seção apresenta o ambiente experimental, o conjunto de amostras utilizado e a configuração dos algoritmos.

7.1. Ambiente Experimental

A Figura 2 apresenta a arquitetura de rede da solução. O ambiente experimental é composto por três servidores físicos Dell PowerEdge 1950 (Intel Xeon, 12 GB de RAM e 270 GB de HDD). Esses servidores são virtualizados com o VMware vSphere 6.5 (três virtualizadores na figura). Sobre essa infraestrutura estão distribuídas três máquinas virtuais (VMs) formando o *cluster* Kafka (virtualizador 3). Também há mais uma VM para um produtor e outra para um consumidor no virtualizador 2. Para os serviços de otimização e monitoramento também há uma VM para cada (virtualizador 1).

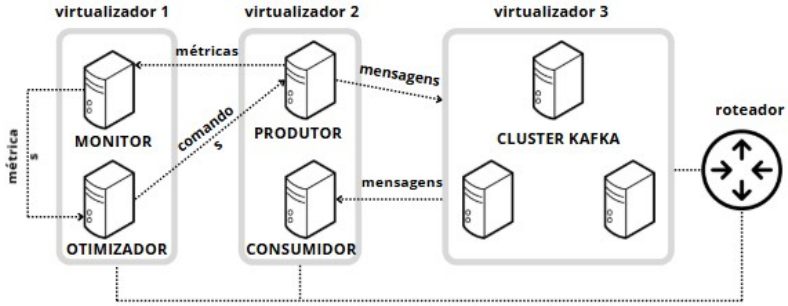

Figura 2. Arquitetura de rede da solução.

Todas as máquinas executam Ubuntu Server 24. Os dados de monitoramento de tráfego de todas as máquinas são enviados para o banco de dados Prometheus. O sistema Grafana é utilizado para a visualização desses dados de métricas. A carga de trabalho é gerada sinteticamente por meio da ferramenta *kafka-producer-perf-test.sh*, com um número fixo de mensagens enviadas uma logo em seguida à outra. Os experimentos, em cada configuração, são realizados em 10 rodadas, enviando 1000 mensagens a cada rodada. Todas as ferramentas empregadas são gratuitas ou de código aberto. A arquitetura de rede e hardware pode ser reproduzida, com os mesmos elementos (VMs), usando equipamentos mais modernos ou em ambiente de nuvem.

7.2. Conjunto de Amostras Experimentais

Para reduzir o número de experimentos, é adotada uma estratégia de planejamento experimental baseada em DOE utilizando o método *Latin Hypercube Sampling* (LHS) [Bao et al. 2018]. Esse método gera amostras heterogêneas em subgrupos com características comuns sobre o espaço de parâmetros, o que garante uma cobertura mais uniforme desse espaço.

Sem o DOE, são necessárias até 2700 execuções (540 combinações em cada um dos cinco diferentes tamanhos de mensagem). Com a abordagem, esse número é reduzido para 300 experimentos, com 60 por tamanho de mensagem, mantendo a cobertura adequada do espaço de busca.

7.3. Configuração dos Algoritmos

Para a modelagem substituta, foi utilizada a biblioteca *XGBoost* em Python, por meio da classe `XGBRegressor`, enquanto a etapa evolutiva foi implementada com a biblioteca *pymoo*, utilizando o algoritmo NSGA-III. Foram treinados modelos independentes para cada objetivo de otimização, tendo como variáveis de entrada os parâmetros *batch.size*, *linger.ms*, *acks*, *compression.type* e o tamanho da mensagem representado por *record_size*. As variáveis categóricas *acks* e *compression.type* foram convertidas para códigos inteiros, permitindo sua utilização no processo de aprendizado. No XGBoost, adotaram-se 300 estimadores, profundidade máxima igual a 6, taxa de aprendizado de 0,05, *subsample* de 0,9, *colsample_bytree* de 0,9, regularização L2 (*reg_lambda*=1,0) e função objetivo `reg:squarederror`. Além disso, os modelos XGBoost foram analisados com separação treino-teste de 75%-25%.

Na otimização multiobjetivo, foram empregados pontos de referência gerados pelo método de Das e Dennis para quatro objetivos (*n_partitions*=12) [Deb and Jain 2013], população de 92 indivíduos, 80 gerações, inicialização aleatória, cruzamento SBX com probabilidade de 0,9 e $\eta = 15$, mutação polinomial com probabilidade de 0,2 e $\eta = 20$, além de semente fixa igual a 42 para reprodutibilidade.

8. Resultados

Nesta seção, são apresentados os resultados obtidos a partir da execução dos experimentos e do processo de otimização. Basicamente, os dados amostrais são lidos, processados dentro da solução e então obtém-se um conjunto de soluções baseado em objetivos.

8.1. Avaliação do Sistema Pré-Otimização

Para analisar o comportamento do Kafka em termos de vazão, latência e variabilidade, as 300 configurações obtidas por meio do DOE são executadas no sistema real antes da etapa de otimização. O resultado dessas execuções gera os dados de treinamento necessários para o XGBoost e para o processamento da população inicial do NSGA-III. Esses conjuntos de dados são uma composição de valores de parâmetros Kafka, o tamanho da mensagem em questão e os valores de métricas de desempenho na entrega de 1000 mensagens do produtor a um *broker*. Para cada configuração, a coleta de dados é realizada 10 vezes, com um intervalo de 5 s entre as execuções.

Com a execução dessas configurações, é possível observar, na Tabela 1, o desempenho por configuração de parâmetros do Kafka para cada tamanho de mensagem e para cada objetivo. Visa-se elencar um conjunto de soluções dentre as quais o decisor possa escolher as soluções ótimas com base em algum critério de pós-processamento.

Com base na Tabela 1, é possível analisar que, no caso das mensagens de 8 kB, por exemplo, a vazão varia com uma diferença superior a 180%. De forma análoga, a latência p95 oscila entre 73,0 e 1997,7 ms, evidenciando variações significativas, considerando o mesmo tamanho de mensagem e diferentes valores de parâmetros. Esse comportamento se repete nos demais tamanhos de mensagem, indicando que o desempenho não é determinado apenas pela carga, mas principalmente pela escolha dos parâmetros de configuração. Já os CVs (variabilidades) reforçam a existência de instabilidade significativa entre as execuções. Em diversas configurações, o CV da latência ultrapassa 1,0, indicando alta dispersão temporal e baixa previsibilidade do sistema. Essa característica

Tabela 1. Melhores e piores resultados para diferentes valores dos parâmetros de configuração do Kafka por tamanho de mensagem.

Tamanho (B)	Vazão (MB/s)		Latência p95 (ms)		CV Vazão		CV Latência	
	Melhor	Pior	Melhor	Pior	Melhor	Pior	Melhor	Pior
1024	**0,878**	0,390	**84,6**	980,3	**0,0309**	0,1579	**0,1128**	0,8248
8192	**5,636**	1,959	**73,0**	1997,7	**0,0643**	0,5027	**0,1538**	1,4558
10240	**5,623**	3,214	**85,1**	910,9	**0,0240**	0,1659	**0,1547**	0,9569
16383	**9,793**	5,102	**74,7**	728,3	**0,0462**	0,2011	**0,1655**	0,8771
65636	**23,932**	9,790	**27,9**	1134,2	**0,0248**	0,2576	**0,2417**	1,5781

é particularmente crítica em aplicações sensíveis a latência, nas quais flutuações podem comprometer acordos de SLA. No caso da vazão, valores elevados de CV indicam inconsistência na taxa de processamento, podendo levar à subutilização de recursos.

Considerando que o NSGA-III é inicializado com os dados da execução das configurações DOE, esses mesmos dados são utilizados tanto para a geração da população inicial quanto para a definição dos pontos de referência no espaço de objetivos. A partir de todos os dados coletados anteriormente, pode-se observar quais configurações de parâmetros levam a melhores desempenhos para cada métrica individualmente. A Tabela 2 evidencia que diferentes objetivos de desempenho levam a configurações distintas nos parâmetros do produtor. Neste exemplo, com um tamanho da mensagem igual a 8 kB, enquanto a maximização da vazão favorece valores elevados de *batch.size* e menor confiabilidade de entrega (*acks*=1), a minimização da latência e da variabilidade leva a configurações com maior controle e, em alguns casos, ao uso de compressão. Esses resultados reforçam o caráter conflitante dos objetivos e evidenciam a inexistência de uma configuração única que seja simultaneamente ótima para todas as métricas, justificando a adoção do algoritmo NSGA-III, pois, a partir desses valores iniciais, a fronteira de Pareto também é inicialmente construída, sendo atualizada com o passar de gerações.

Tabela 2. Melhores configurações de parâmetros para *record_size* de 8 kB considerando diferentes objetivos.

Objetivo	batch.size	acks	linger.ms	compress
Maior vazão	262144	1	0	none
Menor latência	131072	0	5	gzip
Menor CV da vazão	4096	all	10	gzip
Menor CV da latência	2048	1	0	lz4

8.2. Avaliação da Otimização

Foram avaliados 7360 indivíduos, por tamanho de mensagem, com uma população de 92 indivíduos por geração. Com isso, a combinação entre a diversidade inicial proporcionada pelo DOE e a avaliação via modelo preditivo substituto permite acelerar a convergência sem comprometer a qualidade das soluções. Nesse papel, o XGBoost, a partir dos dados de treinamento, fornece os dados relacionados ao desempenho das soluções do NSGA-III. A avaliação dessas soluções em ambiente real poderia levar muito tempo, evidenciando o caráter econômico e prático da solução.

Os parâmetros aplicados no XGBoost buscam equilibrar capacidade preditiva e controle de sobreajuste. Esse risco também foi reduzido pela cobertura amostral via LHS, pela regularização nativa do XGBoost e pela validação final das soluções em ambiente

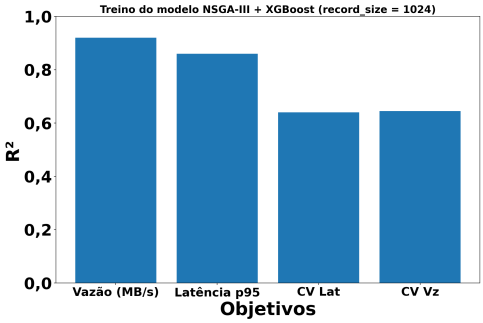

Figura 3. Desempenho do treinamento com record_size em 1 kB.

real. A Figura 3 apresenta o valor do coeficiente de determinação (R^2) calculado em função dos valores preditos pelo XGBoost e os obtidos no ambiente real. Observa-se uma maior capacidade preditiva para métricas centrais de desempenho, como vazão e latência p95, enquanto métricas de variabilidade apresentaram valores moderados de R^2, ou seja, há uma maior complexidade em modelar oscilações temporais do sistema.

Considerando o conjunto de soluções ótimas presente na fronteira de Pareto, é possível destacar os melhores resultados em termos de objetivos. Essas configurações foram submetidas à execução real para comparar seus resultados com os valores estimados usados no modelo, conforme a Figura 4. Observa-se que os valores de vazão estimados apresentam valores próximos aos reais, com pequenas diferenças que se tornam mais evidentes em cenários de maior volume de dados. O aumento da vazão com o crescimento do *record_size* pode ser explicado pela redução do impacto dos *overheads* fixos associados ao envio de mensagens, como serialização, requisições e controle de protocolo. Em mensagens pequenas, esses custos representam uma fração significativa do total transmitido, limitando a eficiência. À medida que o tamanho da mensagem aumenta, tais *overheads* são amortizados por um maior volume de dados úteis, resultando em melhor aproveitamento da largura de banda e maior eficiência de transmissão. Esse efeito é reforçado por mecanismos de *batching*, que reduzem a quantidade de requisições e aumentam a taxa efetiva de envio de dados.

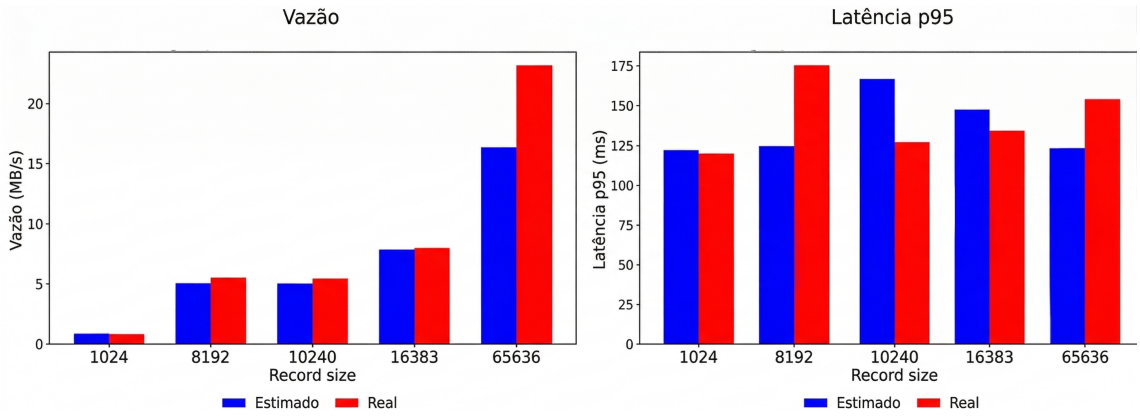

Figura 4. Comparação entre valores estimados e reais para configurações representativas da fronteira de Pareto.

Considerando o fato de haver uma proximidade de valores entre o estimado e o

real, isso mostra que o XGBoost conseguiu eficiência em sua função de estimar os valores dos objetivos das soluções geradas pelo NSGA-III.

A Figura 5 apresenta a relação entre a vazão real e o CV real, diferenciando os pontos pelo tamanho da mensagem. Observa-se a formação de agrupamentos distintos, indicando que a escolha de configurações com menos variabilidade não representa uma vazão significativamente menor. Configurações associadas aos maiores valores de *record_size* tendem a apresentar maiores vazões, mantendo níveis reduzidos de variabilidade.

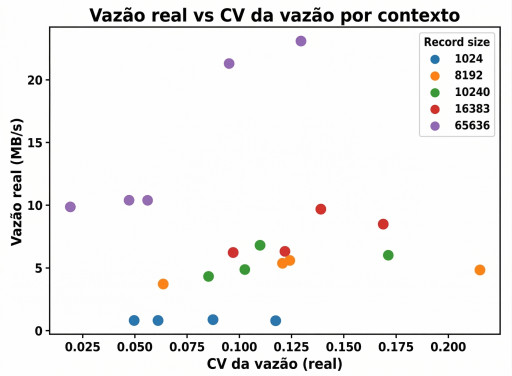

Figura 5. Vazão X CV da vazão.

Já a Figura 6 mostra um recorte dos resultados finais da execução do algoritmo - Executado fronteira Pareto real - quando comparados com os resultados estimados - estimado (NSGA-III + XGBoost) - e com resultados iniciais - Executado DOE. Considerando a vazão e a latência com tamanho de mensagem de 1 kB, é possível observar que em geral as soluções estimadas são mais otimistas do que a execução real nesses critérios de seleção final. Por outro lado, as soluções otimizadas mostraram-se mais eficientes do que as soluções usadas no início (DOE), se não em toda a fronteira de Pareto, em relação a essas métricas, mas ainda assim com indivíduos de melhor desempenho. O fato de a estimativa ter valores mais otimistas que a execução está relacionado aos erros do modelo preditivo, porém foram preservadas as classificações das soluções e a tendência para que a busca pudesse ser guiada.

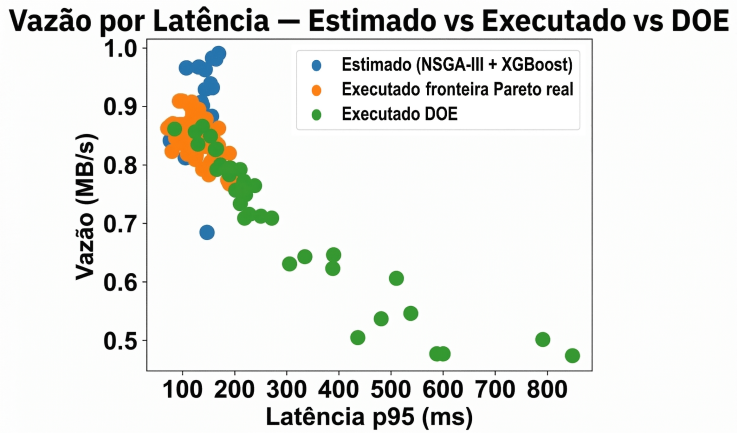

Figura 6. Vazão X Latência com Tamanho de Mensagem igual a 1 kB.

8.3. Comparação entre as Configurações Padrão e Recomendada

A Figura 7 apresenta a comparação entre uma configuração padrão do produtor Kafka (padrão) e uma configuração obtida da fronteira de Pareto (R1). A configuração padrão foi definida com base nos valores recomendados na documentação oficial do Kafka. No caso da configuração de Pareto, ela foi obtida avaliando a melhor relação entre vazão (maior) e o CV da vazão (menor).

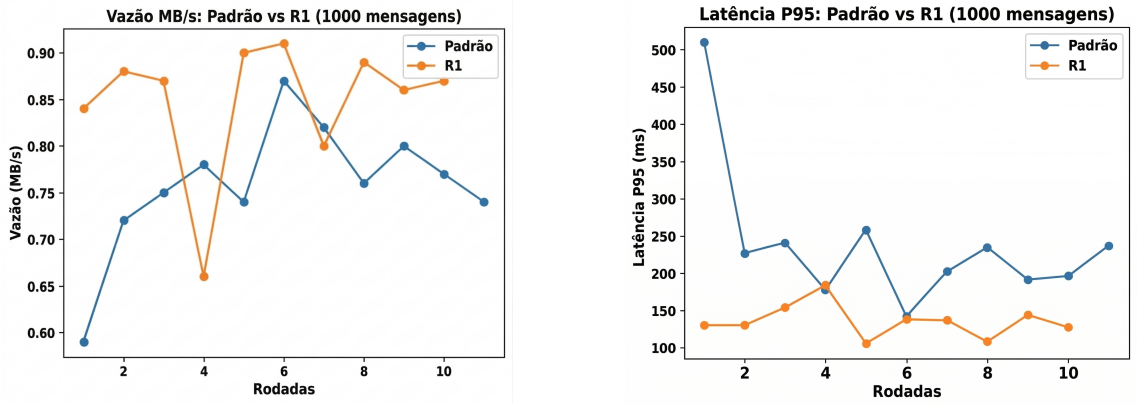

Figura 7. Comparação entre a configuração padrão e a de melhor relação entre vazão e coeficiente da vazão.

Observa-se que, na configuração padrão, a primeira rodada apresenta desempenho significativamente inferior às demais, o que pode ser atribuído a efeitos de aquecimento do sistema (*warm-up*), como a inicialização de *buffers*, caches e a estabilização da comunicação com o *broker*. Esse comportamento é comum em sistemas distribuídos.

Por outro lado, a configuração otimizada apresenta um comportamento mais estável entre as execuções, com menor dispersão dos valores desde as primeiras rodadas, principalmente para a latência. Considerando todas as rodadas, os resultados indicam ganhos expressivos em todas as métricas analisadas. A configuração otimizada apresentou em média aumento de aproximadamente 12% na vazão, redução de 43% na latência p95, diminuição de 8% na variabilidade da vazão e redução de 59% na variabilidade da latência em comparação com a configuração padrão. Os parâmetros selecionados pelo processo de otimização contribuem para uma inicialização mais eficiente e para uma menor sensibilidade a variações transitórias, reforçando a robustez da solução proposta.

9. Conclusão

Este trabalho apresenta uma abordagem para a otimização multiobjetivo da configuração de produtores Apache Kafka. A formulação considera simultaneamente métricas de desempenho e estabilidade, incluindo latência p95, vazão e suas variabilidades. Os resultados evidenciam a complexidade do espaço de configuração, destacando que pequenas variações nos parâmetros podem impactar significativamente o desempenho. Nesse contexto, o modelo substituto mostrou-se essencial para reduzir o custo experimental e viabilizar a exploração eficiente do espaço de busca, conseguindo aproximar valores estimados e reais, especialmente para vazão e latência, enquanto as métricas de variabilidade apresentaram maior dispersão, conforme esperado. Ainda assim, o modelo preserva a relação relativa entre configurações, permitindo ao NSGA-III identificar soluções relevantes. Para

aperfeiçoar a precisão da solução e trabalho futuro, as métricas padrão do XGBoost relacionadas a erros de predição podem ser adotadas na solução, junto com a técnica de curva de aprendizado com validação cruzada.

Agradecimentos

Esta pesquisa é parcialmente financiada pelo CNPq (proc. 405940/2022-0 e 408255/2023-4), CAPES – Código de Financiamento 88887.954253/2024-00 e FAPERJ.

Referências

Apache Software Foundation (2026). Apache kafka producer configuration. `https://kafka.apache.org/42/configuration/producer-configs/`. Acesso em: 15 mar. 2026.

Bao, L., Liu, X., and Chen, W. (2018). Learning-based automatic parameter tuning for big data analytics frameworks. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 181–190. IEEE.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Deb, K. and Jain, H. (2013). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4):577–601.

Dou, H., Zhu, S., Zhou, Y., Zhang, Y., Mei, J., Wu, Y., and Dai, J. (2024). Cotuner: Joint optimization of resource configuration and software parameters for recurring streaming jobs on the cloud. In *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 87–96. IEEE.

Fu, Y. and Soman, C. (2021). Real-time data infrastructure at uber. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2503–2516.

Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70.

Kang, Z., Barve, Y. D., Bao, S., Dubey, A., and Gokhale, A. (2023). Dmsconfig: Automated configuration tuning for distributed iot message systems using deep reinforcement learning. *arXiv preprint arXiv:2302.09146*.

Kreps, J., Narkhede, N., Rao, J., et al. (2011). Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, volume 11, pages 1–7. Athens, Greece.

Ma, H., Ma, Y., Guo, T., Yuan, H., et al. (2025). An automated method for solving configuration of distributed messaging systems: Dmga-pso algorithm. *Expert Systems with Applications*, 277:127161.

Montgomery, D. C. (2017). *Design and analysis of experiments*. John wiley & sons.

Raptis, T. P. and Passarella, A. (2023). A survey on networked data streaming with apache kafka. *IEEE access*.

Zhang, Q. and Li, H. (2007). Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731.