Sistema de Recomendação em Tempo Real via Processamento de Fluxo sobre Bancos de Dados Distribuídos

Lorenzo Ficher[1], Marcus Querol[1], Mirieli Oliveira[1], Natalia Gonçalves[1], Vinícius Gonçalves[1], Yuri Martins[1], Vitor Balsanello[1], Maicon Bernardino[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
97.546-550 – Alegrete – RS – Brasil

{lorenzoficher, marcusquerol, mirielioliveira, nataliaperri, viniciusdsg2, yuriromeiro, vitorbalsanello}.aluno@unipampa.edu.br, bernardino@acm.org

Abstract. *Traditional batch-based recommendation systems fail to adapt to user interactions in real-time. This paper proposes an integrated architecture for recommendation systems that combines stream processing and distributed databases to deliver low-latency personalized suggestions. The solution is based on a hybrid architecture that employs a context-aware collaborative filtering algorithm and an optimized caching layer. The goal is to solve the integration bottleneck between the processing, algorithm, and persistence layers, ensuring the delivery of consistent and up-to-date recommendations in milliseconds.*

Resumo. *Sistemas de recomendação tradicionais baseados em lote falham em se adaptar às interações do usuário em tempo real. Este artigo propõe uma arquitetura integrada para sistemas de recomendação que combina processamento de fluxo e bancos de dados distribuídos para oferecer sugestões personalizadas de baixa latência. A solução se baseia em uma arquitetura híbrida que utiliza um algoritmo de filtragem colaborativa sensível ao contexto e uma camada de caching otimizada. O objetivo é resolver o gargalo na integração entre as camadas de processamento, algoritmo e persistência, garantindo a entrega de recomendações consistentes e atualizadas em milissegundos.*

1. Introdução

A era digital trouxe um volume massivo de dados, tornando necessário o uso de Sistemas de Recomendação (SRs) para que fosse possível conectar usuários a produtos, filmes ou notícias [Barré et al. 2021]. No entanto, os modelos tradicionais, baseados em processamento em lote, apresentam limitações [Barajas and Li 2005]. Sendo assim, os sistemas de recomendação em tempo real surgem como uma evolução essencial, processando dados à medida que são gerados e oferecendo respostas imediatas e personalizadas [Chandramouli et al. 2011].

O avanço de tecnologias de *streaming*, como Apache Kafka, Apache Flink e Spark Streaming, possibilitou o desenvolvimento de sistemas capazes de analisar e reagir a eventos em frações de segundo [Chang et al. 2016]. Por exemplo, o estudo de [Lu et al. 2022] indica que a integração de arquiteturas de fluxo com modelos de aprendizado incremental melhora significativamente a precisão e o tempo de resposta dos SRs.

Ainda assim, há desafios relacionados à manutenção da consistência dos dados, à latência de processamento e ao custo computacional envolvido na atualização contínua dos modelos.

Alguns estudos propõem arquiteturas de atualização incremental que utilizam *buffers* temporários para evitar a degradação da performance do modelo [Mi et al. 2020], enquanto outros exploram o uso de técnicas de aprendizado contínuo para permitir a adaptação dos SRs sem a necessidade de reprocessar todo o histórico de dados [Zhang et al. 2024]. Essas abordagens indicam que a tendência atual é buscar equilíbrio entre desempenho, precisão e custo operacional.

Diante desse cenário, este artigo propõe uma arquitetura para sistemas de recomendação em bancos de dados distribuídos, integrando mecanismos de recomendação em tempo real com tecnologias de processamento de fluxo. Busca-se analisar de que forma essa integração impacta a escalabilidade e a capacidade de resposta desses sistemas. Além disso, o trabalho investiga os principais desafios técnicos associados a essa abordagem, bem como as soluções mais promissoras apontadas na literatura recente.

2. Fundamentação Teórica

2.1. Sistemas de Recomendação

Sistemas de recomendação são tecnologias destinadas a apoiar usuários na descoberta de itens de interesse em meio a grandes volumes de informação. Segundo [Ricci et al. 2015], esses sistemas utilizam algoritmos que analisam o comportamento passado e características de itens ou usuários para gerar recomendações personalizadas. Entre as principais abordagens destacam-se a filtragem colaborativa, que baseia-se em padrões de similaridade entre usuários [Koren et al. 2009], e a filtragem baseada em conteúdo, que considera atributos dos itens para inferir relevância [Lops et al. 2011]. Métodos híbridos e técnicas de aprendizado profundo têm se tornado cada vez mais comuns, melhorando a precisão e a capacidade de generalização [Zhang et al. 2019].

Além disso, a integração desses sistemas com arquiteturas de processamento de fluxo de dados tem se mostrado essencial para permitir recomendações em tempo real, atualizando continuamente os resultados conforme novas interações de usuários e mudanças contextuais ocorrem. A arquitetura proposta, ilustrada na Figura 1, baseia-se na extração de características a partir de textos de usuários. O fluxo de dados utiliza a Modelagem de Representação do Item. Essa integração otimiza a sensibilidade do sistema a eventos recentes, permitindo que as recomendações sejam ajustadas dinamicamente com base em novas interações, mitigando a dependência exclusiva de modelos estáticos baseados em dados históricos.

2.2. Processamento de Fluxo de Dados

O paradigma de processamento de fluxo de dados difere do modelo em lote (*batch processing*), pois permite analisar eventos em tempo real, reduzindo significativamente a latência na geração de *insights* [Akidau et al. 2015]. Ferramentas como Apache Kafka, Apache Flink e Spark Streaming viabilizam a ingestão e processamento contínuo de grandes volumes de dados, suportando aplicações que exigem respostas imediatas, como detecção de fraude, monitoramento de redes sociais e sistemas de recomendação [Zaharia et al. 2016].

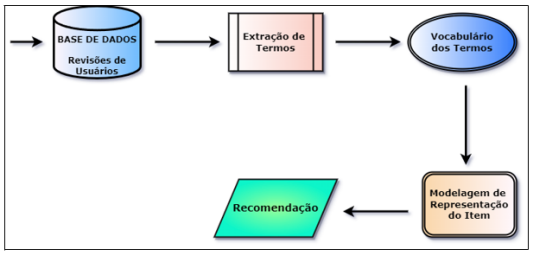

Figura 1. Arquitetura Conceitual de um Sistema de Recomendação

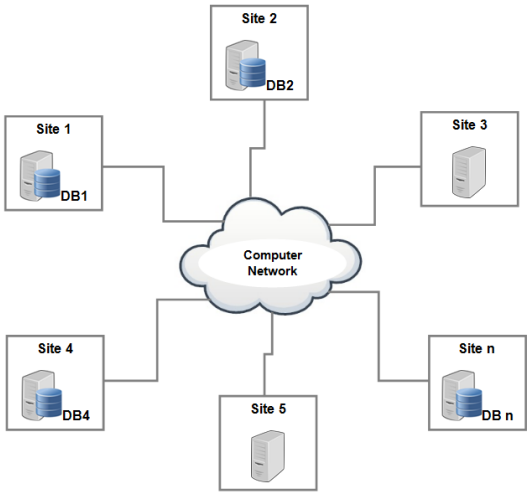

Figura 2. Arquitetura de um SBDD ([Ezéchiel et al. 2019])

No contexto dos SRs, o processamento de fluxo de dados potencializa a capacidade de adaptação instantânea das recomendações, permitindo respostas mais rápidas e contextualizadas. No entanto, essa integração também traz desafios técnicos importantes, como o balanceamento entre latência e precisão, já que a necessidade de respostas imediatas pode reduzir o tempo disponível para análises mais complexas, e o impacto da distribuição dos dados na consistência dos resultados gerados.

2.3. Banco de Dados Distribuídos

Um Sistema de Banco de Dados Distribuído (SBDD) armazena e processa informações em diversos servidores interconectados (nós), surgindo como resposta às limitações de escalabilidade e disponibilidade dos bancos tradicionais. Caracterizam-se pela escalabilidade horizontal, flexibilidade de design *schema-less* e replicação de dados, que aceleram consultas e favorecem aplicações em larga escala, como sistemas de recomendação [Kleppmann 2017]. Seu funcionamento se fundamenta no Teorema CAP [Brewer 2000], que afirma que não se pode assegurar consistência (C), disponibilidade (A) e tolerância a partições (P) ao mesmo tempo. Em tais sistemas, A e P são geralmente priorizados, usando consistência eventual para garantir alta disponibilidade e reduzir o tempo de resposta.

A consistência eventual garante que todas as réplicas se tornem consistentes com o tempo, permitindo respostas rápidas mesmo que dados recentes ainda não tenham se propagado completamente. Embora essa alternativa melhore a escalabilidade e a resiliência a falhas, ela também apresenta dificuldades em relação à sincronização de réplicas e à coordenação de transações [Khandal 2024]. Essa abordagem distribui os dados entre diferentes nós interconectados por uma rede, permitindo maior escalabilidade e tolerância a falhas. A figura 1 ilustra uma arquitetura típica de um sistema de banco de dados distribuído, na qual múltiplos sites armazenam e replicam informações de forma coordenada, garantindo disponibilidade mesmo diante de falhas ou atrasos na propagação das atualizações. Embora a replicação e o monitoramento constantes sejam complexos e custosos, as técnicas de balanceamento de carga e gerenciamento adaptativo de tráfego

ajudam a evitar gargalos e a manter o desempenho. Dessa forma, os SBDD continuam sendo fundamentais para aplicações contemporâneas que demandam alta disponibilidade e baixa latência em situações com milhões de interações simultâneas.

3. Trabalhos Relacionados

A literatura atual sobre sistemas de recomendação, explora a transição para vários tipos de sistema. Nesse contexto, surgiram padrões como a Arquitetura Lambda, que visa unir a robustez do processamento em lote com a velocidade do processamento em tempo real, e a Arquitetura Kappa, que propõe uma abordagem simplificada baseada unicamente em fluxo. Contudo, ambas apresentam desafios em termos de complexidade operacional e consistência de dados, motivando a exploração de novas arquiteturas híbridas [Marz and Warren 2015].

O trabalho seminal de [Sarwar et al. 2001] popularizou a filtragem colaborativa baseada em itens, um método eficiente para dados estáticos, mas que apresenta alta latência e dificuldades com o *concept drift* em ambientes dinâmicos. Para além da simples interação usuário-item, a pesquisa avançou para os Sistemas de Recomendação Sensíveis ao Contexto (CARS), que buscam aumentar a relevância das sugestões ao incorporar variáveis dinâmicas como localização, tempo e atividade do usuário [Adomavicius and Tuzhilin 2011]. Em resposta, a pesquisa recente tem focado em sistemas baseados em processamento de fluxo, como os abordados por [Cai et al. 2020] e [Zaharia et al. 2013], que adaptaram algoritmos de fatoração de matrizes, e [He et al. 2017], que propôs abordagens com redes neurais.

A escalabilidade e a persistência de dados em tempo real são cruciais. A pesquisa de [Lian et al. 2014] trata da eficiência no treinamento de modelos, e a adoção de bancos de dados NoSQL, como Apache Cassandra e Redis, tem sido fundamental para o armazenamento de interações. Trabalhos como o de [Jain et al. 2014] exploram a persistência em sistemas de chave-valor. No entanto, a otimização da integração com sistemas de fluxo vai além do simples armazenamento, abrangendo o desafio de servir dados e características (*features*) para os modelos com latência de milissegundos. Este problema tem impulsionado a pesquisa em *feature stores* em tempo real e em estratégias de *caching* distribuído, áreas nas quais a consistência e a disponibilidade dos dados ainda são desafios ativos [Breck et al. 2017].

Estudo	Algoritmo/Foco	Arquitetura	Processamento	*Caching Features*
Sarwar *et al.*	Filtragem Colaborativa	Lote (Batch)	Lote	Não
Zaharia *et al.*	Plataforma (D-Streams)	Micro-Lote/Fluxo	Fluxo	Não
He *et al.*	Neural CF / Precisão	Lote/Híbrida	Lote/Micro-Lote	Não
Jain *et al.*	Persistência/Chave-Valor	Distribuída (DB)	Híbrido	Persistência
Lian *et al.*	Otimização do Treinamento	Distribuída (Treinamento)	Lote/Micro-Lote	Não
Nossa Proposta	**CF Baseada em Contexto**	**Híbrida Integrada**	**Fluxo Contínuo**	**Sim (Otimização)**

Tabela 1. Comparativo de Abordagens na Literatura para Sistemas de Recomendação em Tempo Real

A análise dos trabalhos relacionados como demonstrados na Tabela 1 revela uma clara trajetória evolutiva, partindo de sistemas em lote [Sarwar et al. 2001] para arquiteturas de fluxo contínuo [Cai et al. 2020, Zaharia et al. 2013]. No entanto, a literatura demonstra que as soluções existentes tendem a otimizar um dos três pilares: algoritmos,

processamento ou persistência, de forma isolada, criando gargalos. Por exemplo, enquanto algoritmos avançados como os CARS [Adomavicius and Tuzhilin 2011] e Redes Neurais [He et al. 2017] melhoram a precisão, sua eficácia em tempo real depende criticamente da disponibilidade de dados atualizados com baixa latência. Este é um desafio que as otimizações de persistência, como as de [Jain et al. 2014], não resolvem completamente, pois o gargalo se desloca para o serviço e a consistência das *features*, como aponta a pesquisa em *feature stores* [Breck et al. 2017].

Portanto, a lacuna na literatura não reside na falta de soluções para cada pilar individualmente, mas sim na ausência de uma arquitetura coesa que otimize a **integração** entre eles. É precisamente nesta interseção que nossa pesquisa se posiciona, propondo uma solução que aborda a entrega de recomendações como um problema de sistema completo, e não como a soma de partes otimizadas separadamente. Para endereçar essa lacuna, a Seção 4 detalha a arquitetura proposta, que se baseia em uma camada de *caching* otimizada e um algoritmo de filtragem colaborativa sensível ao contexto. Esses componentes são projetados para atuar precisamente na interseção crítica entre a persistência (Bancos de Dados Distribuídos) e o processamento (Fluxo de Dados), resolvendo o gargalo de integração de forma específica.

4. Desenvolvimento

4.1. Arquitetura Proposta

Um sistema de recomendação em tempo real sobre bancos de dados distribuídos funciona melhor quando organizado em quatro camadas: ingestão de dados, processamento de fluxo, armazenamento distribuído e entrega das recomendações. A ingestão de dados é o ponto de entrada: tudo que o usuário faz (cliques, buscas ou avaliações) é capturado quase instantaneamente. Para isso, usamos plataformas de *streaming* como Apache Kafka ou similares, que conseguem lidar com grandes volumes de dados sem travar [Chandramouli et al. 2011, Zaharia et al. 2013]. Depois, entra o processamento de fluxo, que pega esses dados em tempo real e transforma em informações úteis para recomendar produtos ou conteúdos. Ferramentas como Spark Streaming ou Flink permitem fazer isso sem deixar a latência subir demais [Cai et al. 2020, Wang et al. 2013].

O armazenamento distribuído garante que todo o histórico de interações do usuário esteja disponível rapidamente e de forma confiável, usando bancos como Cassandra ou HBase [Jain et al. 2014, Kleppmann 2017]. Por fim, a camada de entrega garante que a recomendação chegue para o usuário de forma praticamente instantânea, criando uma experiência fluida, *e.g.* Netflix ou Spotify [Gomez-Uribe and Hunt 2015, Barré et al. 2021]. A Tabela 2 resume o papel de cada camada.

Camada	Papel	Tecnologias
Ingestão	Coleta de eventos gerados pelos usuários (cliques, buscas, avaliações).	Apache Kafka, Apache Pulsar
Processamento	Transformação, agregação e modelagem contínua dos dados em fluxo.	Apache Flink, Spark Streaming
Armazenamento	Persistência e replicação de dados em múltiplos nós distribuídos.	Cassandra, HBase
Entrega	Disponibilização das recomendações atualizadas ao usuário final.	Redis, API REST

Tabela 2. Resumo das camadas da arquitetura proposta

4.2. Fluxo de Dados

O fluxo de dados em sistemas, funciona em quatro etapas: coleta, pré-processamento, modelagem e entrega. Na coleta, cada clique ou avaliação é registrado em tempo real.

O pré-processamento limpa, normaliza e enriquece os dados, garantindo que o modelo de recomendação não fique "confuso" com informações ruins [Barajas and Li 2005]. Depois, o modelo de recomendação processa tudo continuamente, ajustando suas sugestões de acordo com o que o usuário faz de mais recente [Sarwar et al. 2001, He et al. 2017]. Por fim, a entrega faz com que o usuário veja recomendações atualizadas em frações de segundo [Barré et al. 2021]. Arquiteturas de fluxo como a Kappa ajudam a unificar o processamento e evitar redundância entre pipelines de *batch* e *streaming*, tornando tudo mais ágil [Chandramouli et al. 2011].

4.3. Algoritmos Aplicáveis

Diferentes algoritmos podem ser empregados conforme o tipo de dado e a necessidade de atualização. O *ALS* (*Alternating Least Squares*) oferece boa precisão e é passível de paralelização, mas seu custo de processamento é alto em fluxos de grande escala [Sarwar et al. 2001, Lian et al. 2014]. Modelos de *Matrix Factorization* online reduzem esse impacto ao atualizar apenas vetores afetados por novas interações [He et al. 2017]. O uso de *Deep Learning* em *streaming* [Cai et al. 2020] permite capturar padrões complexos e não lineares de comportamento, embora exija hardware especializado (GPU) para manter baixa latência. Já técnicas de *Reinforcement Learning* ajustam recomendações dinamicamente, aprendendo com o *feedback* do usuário, mas podem gerar instabilidade se o modelo não for bem calibrado [Barré et al. 2021]. Os algoritmos adequados variam conforme o cenário, e cada um apresenta vantagens e limitações. A Tabela 3 resume as principais características de cada abordagem, destacando seus pontos fortes e desafios conforme a literatura analisada.

Algoritmo	Vantagens	Limitações
ALS (*Alternating Least Squares*)	Alta precisão e paralelização, adequada para grandes volumes de dados.	Alto custo computacional em fluxos contínuos muito extensos [Sarwar et al. 2001, Lian et al. 2014].
Matrix Factorization Online	Permite atualização incremental sem reprocessar todo o modelo.	Menor estabilidade sob alta variação de entrada [He et al. 2017].
Deep Learning em Streaming	Captura padrões complexos e não lineares em tempo real.	Requer infraestrutura especializada (GPU) e maior consumo de recursos [Cai et al. 2020].
Reinforcement Learning	Adapta-se dinamicamente ao *feedback* do usuário, otimizando recomendações.	Pode se tornar instável se o agente não for bem calibrado [Barré et al. 2021].

Tabela 3. Vantagens e limitações de algoritmos aplicáveis

4.4. Desafios e Reflexões

Apesar da maturidade dessas tecnologias, os desafios permanecem significativos. A escalabilidade é o principal deles, pois os sistemas precisam lidar com milhões de eventos por segundo sem comprometer o desempenho [Chandramouli et al. 2011]. A latência continua sendo um gargalo crítico: quanto mais camadas intermediárias, maior o tempo de resposta [Wang et al. 2013]. A consistência dos dados em sistemas distribuídos, embora mitigada por estratégias como replicação e particionamento, ainda exige *trade-offs* complexos entre disponibilidade e confiabilidade [Jain et al. 2014].

Além disso, o custo operacional de *clusters* de processamento e armazenamento em nuvem pode ser elevado, demandando ajustes finos entre desempenho e orçamento [Cai et al. 2020]. Por outro lado, essas soluções oferecem flexibilidade e personalização em larga escala, demonstrando por que ainda são a base de recomendadores modernos em empresas como Netflix e Amazon [Gomez-Uribe and Hunt 2015].

Desafio	Abordagem / Mitigação na Proposta
Latência	Camada de *Caching* Otimizada (Redis) para leituras em milissegundos, desacoplando o serviço da persistência lenta e do re-treinamento [Jain et al. 2014].
Consistência	Invalidação de cache via fluxo (*stream-driven invalidation*), garantindo que o cache reflita as interações mais recentes processadas pelo Flink [Zaharia et al. 2013].
Escalabilidade	Uso de componentes nativamente distribuídos (Kafka, Flink, Cassandra) que permitem escalabilidade horizontal das camadas de ingestão, processamento e persistência.
Custo de Modelo	Algoritmo de atualização incremental (CF-C), que modifica apenas vetores afetados em vez de reprocessar o *batch* completo, como em abordagens tradicionais [He et al. 2017, Sarwar et al. 2001].
Adaptação	Processamento de fluxo contínuo que permite a adaptação do modelo (CF-C) a cada nova interação, em vez de micro-lotes [Wang et al. 2013].

Tabela 4. Resumo dos desafios e as mitigações propostas pela arquitetura.

4.5. Detalhamento Técnico da Proposta

Uma lacuna identificada na literatura está na falta de uma arquitetura coesa que otimize a integração do processamento, algoritmo e persistência dos pilhares, criando um gargalo de serviço (*feature serving*) bem conhecido[Jain et al. 2014]. Nossa proposta visa esse problema no coração da interseção crítica entre o processamento de fluxo e bancos de dados distribuídos. A fim de entregar recomendações consistentes e atualizadas em milissegundos, a solução combina um Algoritmo de Filtragem Colaborativa Sensível ao Contexto para garantir relevância, e uma Camada de *Caching* Otimizada para garantir latência.

O Algoritmo CF-C, que adapta modelos de fatoração de matrizes ou aprendizado profundo [He et al. 2017] para operar em modo incremental, é vital para manter a relevância em um ambiente dinâmico. Ele absorve e integra as variáveis dinâmicas de contexto (como tempo e atividade do usuário), que são extraídas e enriquecidas pelo motor de Processamento de Fluxo [Zaharia et al. 2013], aumentando a relevância das sugestões para além da simples interação usuário-item [Sarwar et al. 2001]. Para evitar o alto custo computacional do reprocessamento de *batch* em grandes fluxos, o modelo utiliza técnicas de atualização incremental [Wang et al. 2013], modificando apenas os vetores afetados por novas interações em tempo real.

A Camada de *Caching* Otimizada atua como o mecanismo de mitigação do gargalo de serviço (*feature serving*) [Jain et al. 2014], um problema que a persistência em bancos NoSQL não resolve completamente. Utilizando um sistema de chave-valor distribuído (como Redis), o cache armazena os vetores de *embeddings* e as *features* contextuais mais recentes, garantindo que o serviço de entrega de recomendação acesse esses dados com latência mínima. A otimização reside no mecanismo de atualização impulsionado pelo fluxo (*stream-driven invalidation*): após o motor de Processamento de Fluxo recalcular os vetores de usuários e itens, ele imediatamente atualiza (ou invalida) as entradas no cache. Esse processo garante que as recomendações sejam baseadas em dados que refletem a interação mais recente do usuário, resolvendo o compromisso de consistência e disponibilidade inerente aos SBDDs.

Dessa forma, a arquitetura proposta ataca o problema de maneira coesa: a ingestão de eventos e o processamento de fluxos [Zaharia et al. 2013], a atualização do modelo [Wang et al. 2013] coincidem com a persistência SBDD e, em seguida, a atualização do cache em uma cadeia de baixa latência. O sistema pode reagir às ações do usuário em segundos e, ao mesmo tempo, executar recomendações rapidamente em milissegundos, resolvendo o gargalo na integração de processos.

5. Considerações Finais

Este estudo propôs uma arquitetura integrada para sistemas de recomendação em tempo real, baseada na combinação do processamento de fluxo de dados com bancos de dados distribuídos. Essa estratégia busca superar as restrições dos sistemas de recomendação convencionais baseados em processamento em lote, que não conseguem responder prontamente às interações dos usuários, o que compromete a relevância das recomendações [Barajas and Li 2005, Chandramouli et al. 2011].

Ademais, vale salientar que os avanços recentes em ambientes de processamento contínuo (*streaming*) representam uma oportunidade estratégica para os sistemas de recomendação se tornarem mais responsivos e adaptativos. Por exemplo, estudos recentes apontam que o uso de janelas deslizantes em fluxos de dados (*"sliding window"*) pode melhorar tanto a atemporalidade quanto a precisão das recomendações ao captar mudanças rápidas no comportamento dos usuários e no contexto de consumo [Liang et al. 2024].

Conforme análises de revisões sistemáticas, desafios clássicos como escalabilidade, *data sparsity* e latência, que permanecem como argumentos centrais para pesquisas e aplicações práticas na área de recomendadores em tempo real [Roy et al. 2022]. Assim, recomenda-se que trabalhos futuros explorem arquiteturas híbridas combinando filtragem colaborativa, aprendizagem profunda e processamento de fluxo de dados, de modo a equilibrar a latência e qualidade da recomendação, bem como garantir a consistência dos resultados em sistemas distribuídos. O uso de janelas deslizantes permite ao sistema capturar padrões de comportamento em curtos intervalos temporais, ajustando dinamicamente a relevância das recomendações à medida que novos eventos são processados.

Por fim, é importante ressaltar que a arquitetura de cache sugerida não é apenas uma resposta à latência, mas também um facilitador essencial para os trabalhos futuros citados. Modelos complexos de *deep learning*, como a Filtragem Colaborativa Neural [He et al. 2017], que exigem alto custo computacional para inferência, tornam-se viáveis para serviço em tempo real justamente porque seus *outputs*, como vetores de estado do usuário, podem ser pré-calculados de forma incremental pelo fluxo [Wang et al. 2013] e armazenados nessa camada de cache de baixa latência, ficando prontos para uso imediato.

Assim, a integração entre processamento contínuo, aprendizado incremental e *caching* distribuído representa uma contribuição significativa para a evolução de arquiteturas escaláveis e sensíveis ao contexto, abrindo novas oportunidades de pesquisa e aplicação em domínios como *streaming* de vídeo, IoT e plataformas de e-commerce.

Referências

[Adomavicius and Tuzhilin 2011] Adomavicius, G. and Tuzhilin, A. (2011). Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer.

[Akidau et al. 2015] Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernandez-Moctezuma, R. J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., and Whittle, S. (2015). The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803.

[Barajas and Li 2005] Barajas, J. M. and Li, X. (2005). Collaborative filtering on data streams. In Jorge, A. M., Torgo, L., Brazdil, P., Camacho, R., and Gama, J., editors, *Knowledge Discovery in Databases*, pages 429–436. Springer Berlin Heidelberg.

[Barré et al. 2021] Barré, A., Al-Ghossein, M., and Abdessalem, T. (2021). A survey on stream-based recommender systems. *ACM Computing Surveys*.

[Breck et al. 2017] Breck, E., Holt, G., Sculley, D., and Causmaecker, P. (2017). The ml test score: A rubric for ml production readiness and technical debt. In *IEEE International Conference on Big Data*, pages 1123–1132. IEEE.

[Brewer 2000] Brewer, E. A. (2000). Towards robust distributed systems (abstract). In *19th Annual ACM Symposium on Principles of Distributed Computing*, page 7, New York, USA. ACM.

[Cai et al. 2020] Cai, C., Ren, Y., Chen, Z., Chen, K., Cui, Z., and Gao, Y. (2020). Distributed real-time recommender system based on shared-nothing architecture. *Journal of Ambient Intelligence and Humanized Computing*, 11(10):4065–4075.

[Chandramouli et al. 2011] Chandramouli, B., Levandoski, J. J., Eldawy, A., and Mokbel, M. F. (2011). Streamrec: A real-time recommender system. In *ACM SIGMOD International Conference on Management of Data*.

[Chang et al. 2016] Chang, S., Zhang, Y., Tang, J., Yin, D., Chang, Y., and Hasegawa-Johnson, M. A. (2016). Streaming recommender systems. In *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[Ezéchiel et al. 2019] Ezéchiel, K., Kant, S., and Agarwal, D. (2019). A systematic review on distributed databases systems and their techniques. *Journal of Theoretical and Applied Information Technology*, 96.

[Gomez-Uribe and Hunt 2015] Gomez-Uribe, C. A. and Hunt, N. (2015). The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19.

[He et al. 2017] He, X., Zhang, H., Kan, M.-Y., and Chua, T.-S. (2017). Neural collaborative filtering. In *Proc. of the 26th International Conference on World Wide Web (WWW)*, pages 173–182. ACM.

[Jain et al. 2014] Jain, P., Kumar, R., and Gupta, G. (2014). A real-time recommender system based on distributed key-value store. In *Proc. of the International Conference on Big Data*, pages 48–53. IEEE.

[Khandal 2024] Khandal, R. (2024). Distributed database systems: Issues in concurrency control, replication and fault tolerance. *ShodhKosh: Journal of Visual and Performing Arts*, 5(7).

[Kleppmann 2017] Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. "O'Reilly Media, Inc.".

[Koren et al. 2009] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

[Lian et al. 2014] Lian, X.-R., Liu, Y., Sun, Y., Wang, R., and Xu, J.-L. (2014). Parallelized stochastic gradient descent with mini-batching for large-scale recommender systems. In *Proc. of the International Conference on Machine Learning*.

[Liang et al. 2024] Liang, J., Zhao, J., Chen, Z., Zhou, L., Li, Y., and Wang, H. (2024). Ensure timeliness and accuracy: A novel sliding window data stream paradigm for live streaming recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 36(5):1556–1570.

[Lops et al. 2011] Lops, P., de Gemmis, M., and Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, pages 73–105. Springer.

[Lu et al. 2022] Lu, P., Yue, Y., Yuan, L., and Zhang, Y. (2022). Autoflow: Hotspot-aware, dynamic load balancing for distributed stream processing. In Lai, Y., Wang, T., Jiang, M., Xu, G., Liang, W., and Castiglione, A., editors, *Algorithms and Architectures for Parallel Processing*, pages 133–151, Cham. Springer International Publishing.

[Marz and Warren 2015] Marz, N. and Warren, J. (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications.

[Mi et al. 2020] Mi, F., Du, X., and Schuller, B. (2020). ADER: Adaptively distilled exemplar replay towards continual learning for session-based recommendation. In *Proc. of the 29th ACM International Conference on Information and Knowledge Management*.

[Ricci et al. 2015] Ricci, F., Rokach, L., and Shapira, B., editors (2015). *Recommender Systems Handbook*. Springer, 2 edition.

[Roy et al. 2022] Roy, S., Sarker, S., Ghosh, A., Sarkar, S., and Kim, K. M. (2022). A systematic review and research perspective on recommender systems. *Journal of Big Data*, 9(1):1–47.

[Sarwar et al. 2001] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. T. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM.

[Wang et al. 2013] Wang, Y., Zhang, Y., Yin, Y., Yi, D., and Wei, B. (2013). A cluster-based incremental recommendation algorithm on stream processing architecture. In *Proc. of the 15th International Conference on Asian Digital Libraries*.

[Zaharia et al. 2013] Zaharia, M., Das, T., Li, H., Shenker, S., and Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. In *Proc. of the 24th Symposium on Operating Systems Principles*.

[Zaharia et al. 2016] Zaharia, M., Xin, R., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016). Apache spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

[Zhang et al. 2019] Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019). Deep learning based recommender system: A survey. *ACM Computing Surveys*, 52(1):1–38.

[Zhang et al. 2024] Zhang, X., Chen, Y., Ma, C., Fang, Y., and King, I. (2024). Influential exemplar replay for incremental learning in recommender systems. In *Proc. of the 38th AAAI Conference on Artificial Intelligence*. AAAI Press.