Jurnal Penelitian Teknik
Volume 2, Nomor 2, Desember 2025: 248-255
E-ISSN: 3088-9596

PERANCANGAN ARSITEKTUR HYBRID RECOMMENDER SYSTEM BERBASIS RUST UNTUK OPTIMALISASI LAYANAN PLATFORM DIGITAL: STUDI KASUS PERANCANGAN PLATFORM PADA PT XYZ

Alindra Putra
[1]Sistem Informasi, Universitas Pamulang, Tangerang, Indonesia, 15310
e-mail: alindrapu@gmail.com[1]

Abstract

The exponential growth of digital interactions on the PT XYZ platform requires a recommendation system capable of handling high throughput with low latency to maintain user engagement. This study aims to design a high-performance Hybrid Recommender System architecture that combines Content-Based and Collaborative Filtering using a Weighted Hybrid mechanism. The design methodology adopts the Rust programming language to address performance instability issues caused by Garbage Collection pauses in traditional backend architectures. The proposed system integrates an asynchronous runtime for handling concurrency, gRPC protocol for efficient inter-service communication, and a hybrid storage strategy utilizing in-memory caching and vector databases. The result of this research is a conceptual architectural framework that leverages Rust's unique memory ownership model to theoretically eliminate stop-the-world execution pauses and ensure memory safety without runtime overhead. This study contributes to the field of software engineering by providing a system design that offers deterministic latency and resource efficiency for real-time recommendation services in large-scale environments.

Keywords: Hybrid Recommender System; Rust; Microservices; Real-time Inference; Backend Architecture

Abstrak

Pertumbuhan eksponensial interaksi digital pada platform PT XYZ menuntut adanya sistem rekomendasi yang mampu menangani *throughput* tinggi dengan latensi rendah untuk menjaga keterlibatan pengguna. Penelitian ini bertujuan untuk merancang arsitektur Sistem Rekomendasi *Hybrid* berkinerja tinggi yang menggabungkan metode *Content-Based* dan *Collaborative Filtering* menggunakan mekanisme *Weighted Hybrid*. Metodologi perancangan mengadopsi bahasa pemrograman Rust untuk mengatasi masalah ketidakstabilan performa yang disebabkan oleh jeda *Garbage Collection* pada arsitektur *backend* konvensional. Sistem yang diusulkan mengintegrasikan *runtime* asinkron untuk menangani konkurensi, protokol gRPC untuk komunikasi antar layanan yang efisien, serta strategi penyimpanan hibrida yang memanfaatkan *cache* memori dan basis data vektor. Hasil dari penelitian ini adalah kerangka kerja arsitektural konseptual yang memanfaatkan model kepemilikan memori Rust untuk secara teoritis menghilangkan jeda eksekusi dan menjamin keamanan memori tanpa *overhead runtime*. Studi ini berkontribusi pada bidang rekayasa perangkat lunak dengan menyediakan desain sistem yang

menawarkan latensi deterministik dan efisiensi sumber daya untuk layanan rekomendasi *real-time* di lingkungan berskala besar.
Keywords: Sistem Rekomendasi Hybrid; Rust; Microservices; Inferensi Real-time; Arsitektur Backend

Pendahuluan

Saat ini, kita hidup di era di mana informasi membanjiri berbagai platform digital secara masif. Kondisi ini sering kali membuat pengguna merasa kewalahan (information *overload*) saat harus memilih konten yang relevan bagi mereka. Di sinilah peran vital sistem rekomendasi. Seperti dijelaskan oleh Aggarwal (2016), sistem rekomendasi bertugas memprediksi apa yang mungkin disukai pengguna dan menyajikannya secara personal. Fitur ini bukan sekadar pelengkap, melainkan komponen kunci bagi banyak aplikasi untuk menjaga agar pengguna tetap betah (*user retention*) dan aktif berinteraksi di dalam platform.

Dalam dunia pengembangan sistem rekomendasi, kita mengenal dua pendekatan klasik: *Content-Based Filtering* dan *Collaborative Filtering*. Pazzani & Billsus (2007) menjelaskan bahwa *Content-Based Filtering* bekerja dengan melihat kemiripan ciri-ciri (*features*) dari item yang pernah kita sukai sebelumnya. Sementara itu, *Collaborative Filtering* bekerja dengan melihat pola komunitas; jika banyak orang yang mirip dengan kita menyukai suatu item, sistem akan merekomendasikannya kepada kita (Su & Khoshgoftaar, 2009). Sayangnya, kedua metode ini punya kelemahan jika berdiri sendiri. *Collaborative Filtering* sering mati kutu saat menghadapi pengguna baru atau item baru yang belum ada datanya (*cold start problem*). Sebaliknya, *Content-Based* cenderung monoton karena hanya merekomendasikan hal yang mirip-mirip saja, tanpa memberikan kejutan atau hal baru (*serendipity*). Solusi yang paling umum diambil oleh industri saat ini adalah menggabungkan keduanya menjadi sistem *Hybrid* untuk saling menutupi kekurangan tersebut (Burke, 2002).

Namun, tantangan membangun sistem *Hybrid* ini bukan cuma soal matematika atau algoritma, tapi juga soal performa sistem di belakang layar (*backend*). Pada platform skala menengah ke atas seperti studi kasus di PT XYZ, sistem dituntut untuk melayani ribuan permintaan per detik dengan sangat cepat. Masalahnya, banyak sistem *backend* konvensional dibangun dengan bahasa pemrograman yang menggunakan manajemen memori otomatis atau *Garbage Collector* (GC). Riset menunjukkan bahwa proses pembersihan memori oleh GC ini terkadang bisa membuat sistem "cegukan" atau berhenti sejenak (*stop-the-world pauses*) yang sulit diprediksi (Jones et al., 2016). Dalam aplikasi *real-time*, jeda sekian milidetik pun bisa mengganggu kenyamanan pengguna.

Untuk menjawab tantangan performa ini, bahasa pemrograman Rust mulai dilirik sebagai solusi alternatif. Rust menawarkan cara unik dalam mengelola memori lewat konsep kepemilikan (*ownership*), yang menjamin aplikasi aman dari kebocoran memori tanpa perlu adanya *Garbage Collector* (Klabnik & Nichols, 2018). Secara teori, ini membuat performa aplikasi jadi jauh lebih stabil dan efisien. Sayangnya, meski Rust sudah populer di kalangan pengembang sistem operasi, literatur yang membahas penggunaannya untuk arsitektur sistem rekomendasi masih sangat jarang. Kebanyakan riset yang ada saat ini masih berkutat pada implementasi model *Machine*

Learning menggunakan Python atau Java, dan belum banyak yang menyentuh aspek arsitektur sistem *high-performance* menggunakan Rust.

Oleh karena itu, penelitian ini hadir untuk mengisi celah tersebut. Kami bertujuan merancang sebuah arsitektur sistem rekomendasi *Hybrid* (gabungan *Content-Based* dan *Collaborative*) yang memanfaatkan ketangguhan bahasa Rust. Menggunakan PT XYZ sebagai studi kasus perancangan, penelitian ini akan fokus pada bagaimana menyusun arsitektur *backend* yang efisien, aman, dan skalabel. Kontribusi utama dari jurnal ini adalah desain arsitektural dan analisis konseptual tentang bagaimana Rust bisa meningkatkan efisiensi sistem rekomendasi, bukan sekadar laporan angka benchmark atau eksperimen algoritma semata.

Metode Penelitian

Penelitian ini menggunakan pendekatan *Design Science Research* (DSR) untuk merancang sebuah arsitektur perangkat lunak. Seperti dijelaskan oleh Hevner et al. (2004), DSR adalah metode yang fokus pada penciptaan solusi teknologi (artefak) untuk menyelesaikan masalah nyata di lapangan. Pendekatan ini dipilih karena tujuan utama kami bukan sekadar mengamati fenomena, melainkan merancang sistem backend yang mampu menangani beban kerja tinggi. Alur penelitian ini dibagi menjadi tiga tahapan logis: analisis kebutuhan, perancangan arsitektur, dan evaluasi desain.

Analisis Kebutuhan Sistem

Tahap awal dimulai dengan memetakan apa saja yang sebenarnya dibutuhkan oleh sistem, dengan mengambil studi kasus pada platform PT XYZ. Proses ini mengacu pada prinsip *Requirements Engineering* agar spesifikasi yang didapat benar-benar presisi (Sommerville, 2011).

Kebutuhan Fungsional: Sistem harus bisa menangani dua jenis data sekaligus: profil konten untuk Content-Based Filtering (CB) dan riwayat interaksi pengguna untuk Collaborative Filtering (CF). Tantangan utamanya adalah bagaimana menggabungkan kedua metode ini secara mulus untuk menghasilkan satu daftar rekomendasi yang akurat bagi pengguna.

Kebutuhan Non-Fungsional: Mengingat skala trafik PT XYZ yang cukup besar, sistem harus memiliki respon yang sangat cepat (*low latency*). Selain itu, sistem harus aman dari kebocoran memori (*memory leaks*) tanpa harus bergantung pada *Garbage Collection* (GC). Seperti diketahui dalam studi Jung et al. (2017), jeda akibat proses pembersihan memori otomatis (*GC pauses*) sering kali menjadi penyebab performa sistem *real-time* menjadi tidak stabil.

Strategi Perancangan Arsitektur

Pada tahap ini, fokus utama adalah merancang struktur backend menggunakan ekosistem Rust. Ada dua strategi kunci yang kami terapkan:

Mekanisme Weighted Hybrid: Untuk menggabungkan kedua algoritma rekomendasi, kami memilih metode *Weighted Hybrid*. Menurut Burke (2002), 250dalah ini 250dalah cara paling efektif untuk mengambil keunggulan dari beberapa metode sekaligus. Secara sederhana, skor rekomendasi akhir (S_{total}) didapatkan dengan menjumlahkan skor dari CB dan CF yang masing-masing sudah diberi bobot, seperti terlihat pada **Persamaan (1)**:

$$S_{total}(u,i) = \alpha \cdot S_{CB}(u,i) + (1-\alpha) \cdot S_{CF}(u,i)$$
Persamaan (1)

Nilai α di sini berfungsi sebagai pengatur; seberapa besar kkita ingin mempercayai hasil dari *Content-Based* dibandingkan *Collaborative filtering*. Arsitektur ini dirancang agar perhitungan kedua skor tersebut berjalan secara paralel (bersamaan) memanfaatkan fitur concurrency di Rust yang aman, sehingga prosesnya jauh lebih cepat.

Model Inferensi *Real-time:* Kami memutuskan untuk tidak menggunakan sistem *batch* (proses tunda semalaman), melainkan *Real-time Inference*. Gomez-Uribe dan Hunt (2015) menekankan bahwa dalam sistem rekomendasi modern, data yang "segar" sangatlah penting. Artinya, jika pengguna menyukai sebuah barang detik ini, rekomendasi berikutnya harus langsung berubah menyesuaikan minat baru tersebut. Di sinilah peran Rust sangat krusial, karena bahasa ini mampu melakukan perhitungan matriks dan penyaringan data di memori dengan sangat efisien tanpa membebani server.

Evaluasi Desain

Tahap terakhir adalah mengevaluasi apakah desain yang diusulkan ini benar-benar lebih baik. Karena penelitian ini bersifat perancangan (desain), evaluasinya dilakukan melalui analisis perbandingan, sebuah metode yang valid dalam DSR menurut Peffers et al. (2007). Kami akan membedah bagaimana Rust menangani memori—terutama konsep kepemilikan data (*ownership*)—dan membandingkannya dengan cara kerja bahasa pemrograman lain yang menggunakan *Garbage Collector*. Dari analisis ini, kita bisa melihat potensi penghematan sumber daya dan peningkatan stabilitas sistem secara teoritis.

HASIL DAN PEMBAHASAN

Bagian ini menyajikan artefak utama hasil penelitian berupa rancangan arsitektur sistem rekomendasi Hybrid. Fokus pembahasan meliputi topologi sistem, spesifikasi teknis komponen utama, serta analisis teoretis mengenai implikasi penggunaan teknologi terpilih terhadap kinerja sistem secara keseluruhan.

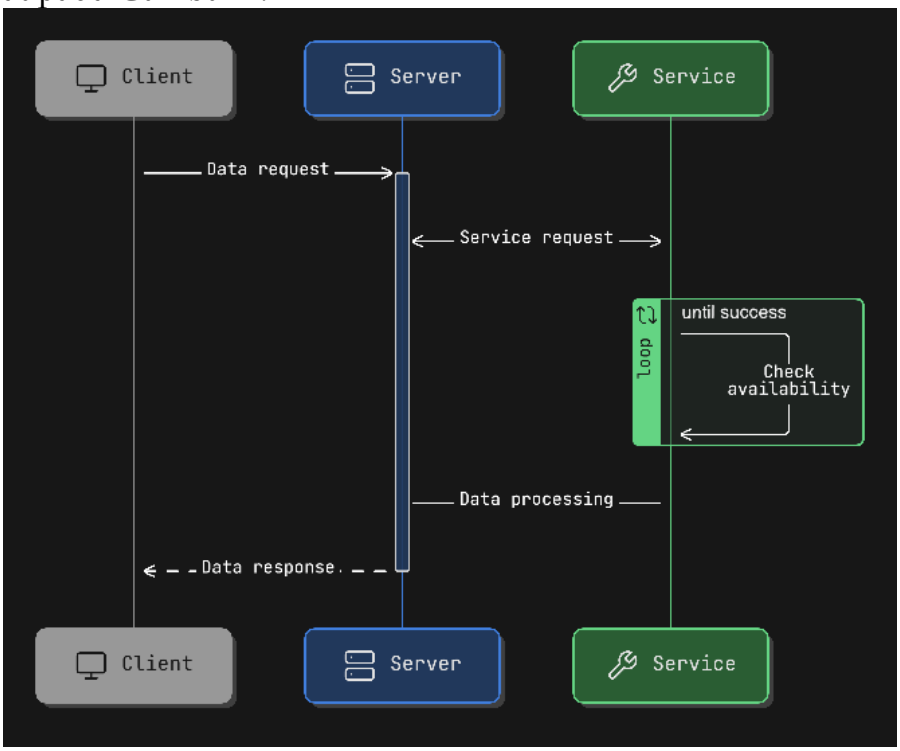
Arsitektur Sistem yang Diusulkan

Berdasarkan analisis kebutuhan, kami merancang arsitektur dengan pola *Microservices* yang terpisah dari aplikasi utama (monolit) PT XYZ. Pemisahan ini bertujuan agar beban komputasi rekomendasi yang berat tidak mengganggu operasional fitur lain.

Secara garis besar, alur kerja sistem dirancang sebagai berikut:

1. ***Ingestion Layer* (Lapisan Masukan):** Menangkap data interaksi pengguna secara *real-time* dan mengirimkannya ke layanan rekomendasi.
2. ***Processing Layer* (Lapisan Pemrosesan):** Inti dari sistem yang dibangun menggunakan Rust. Di sini, logika *Content-Based* dan *Collaborative Filtering* dijalankan secara paralel.
3. ***Data Layer* (Lapisan Data):** Menggunakan strategi penyimanan Hybrid (Redis dan Vector Database) untuk melayani permintaan data dengan kecepatan tinggi.

Visualisasi lengkap mengenai topologi sistem dan interaksi antar-komonen tersebut dapat dilihat pada **Gambar 1**.



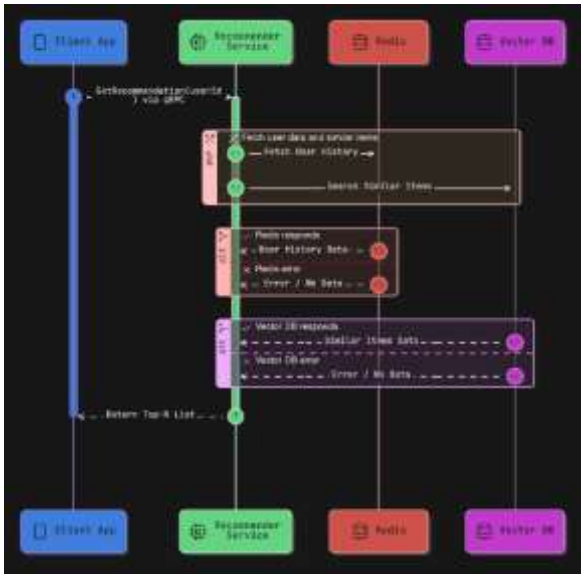
Gambar 1. Arsitektur Sistem Rekomendasi Hybrid yang Diusulkan. Sistem menggunakan protokol gRPC untuk komunikasi dan manajemen memori berbasis Rust untuk pemrosesan paralel.

Detail Komponen Teknis

Keunggulan arsitektur ini terletak pada pemilihan komponen teknologi yang spesifik untuk menangani masalah latensi dan konkurensi. Berikut adalah detail dari tiga keputusan teknis utama:

Protokol Komunikasi: gRPC Untuk komunikasi antar-layanan (*backend-to-backend*), desain ini meninggalkan REST API konvensional dan beralih menggunakan gRPC. Keputusan ini didasarkan pada kebutuhan akan efisiensi transmisi data. Indrasiri dan Kuruppu (2017) menjelaskan bahwa gRPC menggunakan *Protocol Buffers* yang merupakan format biner, sehingga ukuran pesan yang dikirim jauh lebih kecil dan proses *serialization/deserialization*-nya lebih cepat dibandingkan format teks JSON. Dalam konteks sistem rekomendasi yang harus mengirimkan daftar ratusan ID produk dalam hitungan milidetik, penggunaan gRPC secara teoritis dapat memangkas *overhead* jaringan secara signifikan.

Manajemen Konkurensi: Asynchronous Runtime (Tokio) Mengingat Rust tidak memiliki *runtime* bawaan untuk operasi asinkron (*async*), arsitektur ini mengadopsi **Tokio** sebagai fondasi eksekusi. Tokio dipilih karena kemampuannya menerapkan model *event-driven non-blocking I/O*. Klabnik dan Nichols (2019) menekankan bahwa pendekatan asinkron di Rust ini sangat efisien memori (*zero-cost abstractions*) dibandingkan model *multi-threading* tradisional. Sistem dirancang untuk melakukan pengambilan data (*data fetching*) ke beberapa sumber data secara bersamaan, bukan berurutan. Alur eksekusi paralel ini, yang bertujuan meminimalkan total waktu tunggu (*latency*), divisualisasikan pada **Gambar**



Gambar 2. Sequence Diagram Alur Inferensi. Diagram menunjukkan eksekusi paralel pada pengambilan data riwayat pengguna (Redis) dan pencarian vektor (Vector DB), yang meminimalkan total latensi sistem.

Strategi Penyimpanan Data: Redis dan Vector Database Untuk mendukung inferensi real-time, arsitektur ini menggunakan kombinasi dua jenis basis data:

Redis (In-Memory Cache): Digunakan untuk menyimpan data yang sering diakses ("data panas") seperti profil pengguna aktif dan riwayat interaksi terakhir. Carlson (2013) menyatakan bahwa penyimpanan berbasis memori seperti Redis mampu memberikan waktu respons sub-milidetik, yang krusial untuk mengambil data input bagi algoritma Collaborative Filtering.

Vector Database: Digunakan untuk komponen Content-Based Filtering. Fitur produk dikonversi menjadi representasi vektor (embeddings) dan disimpan di sini. Menurut Malkov dan Yashunin (2018), penggunaan struktur data khusus seperti HNSW (Hierarchical Navigable Small World) dalam basis data vektor memungkinkan pencarian kemiripan (similarity search) pada data berdimensi tinggi dilakukan jauh lebih cepat daripada pemindaian database relasional biasa.

Analisis dan Pembahasan

Pada bagian ini, kita membedah mengapa desain di atas secara teoritis lebih unggul dibandingkan arsitektur konvensional yang menggunakan bahasa berbasis Garbage Collector (GC) seperti Java atau Python.

Eliminasi Jeda Eksekusi (Stop-the-World) Masalah klasik pada sistem rekomendasi berskala besar adalah ketidakstabilan latensi. Pada sistem berbasis Java atau Python, sesekali sistem akan mengalami "cegukan" atau jeda singkat saat Garbage Collector membersihkan memori yang tidak terpakai. Jones et al. (2016) mencatat bahwa jeda ini bisa berlangsung dari beberapa milidetik hingga detik, yang fatal bagi pengalaman pengguna real-time. Dengan Rust, manajemen memori ditangani saat kompilasi melalui sistem Ownership. Artinya, tidak ada proses pembersihan memori di latar belakang saat aplikasi berjalan. Desain ini menjamin bahwa waktu yang dibutuhkan untuk menghasilkan rekomendasi akan selalu stabil (deterministik).

Efisiensi Sumber Daya Melalui *Zero-Cost Abstractions* Penggunaan Tokio dan gRPC dalam ekosistem Rust memungkinkan apa yang disebut sebagai *Zero-Cost Abstractions*. Artinya, pengembang bisa menulis kode tingkat tinggi yang mudah dibaca, namun saat dijalankan, kode tersebut dikompilasi menjadi instruksi mesin yang seefisien kode rakitan manual. Secara implikasi desain, ini berarti PT XYZ dapat melayani jumlah pengguna yang sama dengan spesifikasi server yang lebih rendah dibandingkan jika menggunakan arsitektur konvensional, menghasilkan efisiensi biaya infrastruktur jangka panjang.

KESIMPULAN

Penelitian ini berhasil merancang sebuah kerangka arsitektur sistem rekomendasi *Hybrid* yang dioptimalkan untuk lingkungan produksi berskala menengah hingga besar. Dengan menggabungkan pendekatan *Content-Based* dan *Collaborative Filtering* menggunakan metode *Weighted Hybrid*, desain ini menawarkan keseimbangan antara relevansi personal dan eksplorasi konten baru. Kontribusi utama dari penelitian ini terletak pada pemanfaatan karakteristik unik bahasa pemrograman Rust dalam lapisan *backend*.

Secara teoretis, analisis arsitektural menunjukkan bahwa penggantian mekanisme manajemen memori berbasis *Garbage Collector* dengan model kepemilikan (*ownership*) Rust mampu mengeliminasi masalah jeda eksekusi (*stop-the-world pauses*). Hal ini memberikan jaminan latensi yang lebih deterministik bagi aplikasi *real-time*. Selain itu, penerapan teknologi pendukung seperti protokol gRPC untuk komunikasi antar-layanan, serta *runtime* asinkron Tokio untuk manajemen konkurensi, memungkinkan sistem untuk memproses ribuan permintaan inferensi secara paralel dengan jejak memori (*memory footprint*) yang jauh lebih efisien dibandingkan arsitektur konvensional.

Meskipun desain ini menjanjikan efisiensi tinggi secara konseptual, validasi empiris melalui implementasi kode produksi belum tercakup dalam lingkup penelitian ini. Oleh karena itu, penelitian selanjutnya disarankan untuk berfokus pada pengembangan prototipe fungsional dan melakukan pengujian beban (*load testing*) untuk mengukur metrik performa aktual, seperti *throughput* per detik dan penggunaan CPU, guna memverifikasi hipotesis efisiensi yang telah diajukan.

REFERENCES

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Springer.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4), 331–370. https://doi.org/10.1023/A:1021240730564

Carlson, J. L. (2013). *Redis in Action*. Manning Publications.

Haryadi, R. N., Rojali, A., & Fauzan, M. (2021). Sosialisasi Penggunaan Online Shop berbasis Website di UMKM Cimanggis. *Jurnal Pengabdian Masyarakat Madani (JPMM)*, *1*(1), 10-16.

Indrasiri, K., & Kuruppu, D. (2017). *gRPC: Up and Running*. O'Reilly Media.

Jones, R., Hosking, A., & Moss, E. (2016). *The Garbage Collection Handbook: The Art of Automatic Memory Management*. CRC Press.

Jones, R., Hosking, A., & Moss, E. (2016). *The Garbage Collection Handbook: The Art of*

Automatic Memory Management. CRC Press.
Klabnik, S., & Nichols, C. (2018). *The Rust Programming Language*. No Starch Press.
Klabnik, S., & Nichols, C. (2019). *The Rust Programming Language* (2nd ed.). No Starch Press.
Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4), 824-836.
Pazzani, M. J., & Billsus, D. (2007). Content-based recommendation systems. In P. Brusilovsky, A. Kobsa, & W. Nejdl (Eds.), *The Adaptive Web* (pp. 325-341). Springer.
Pugu, M. R., Riyanto, S., & Haryadi, R. N. (2024). *Metodologi Penelitian; Konsep, Strategi, dan Aplikasi*. PT. Sonpedia Publishing Indonesia.
Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009, Article ID 421425. https://doi.org/10.1155/2009/421425