Balanced B-Trees untuk Optimalisasi *Query* pada Basis Data Graf : Sebuah Kajian

Wina Witanti[1], Fildzah Syarafina[2]
[1,2]Program Studi Informastika Fakultas Sains dan Informatika
Universitas Jenderal Achmad Yani
Cimahi, Indonesia
e-mail: [1]witanti@gmailcom , [2]fildzahsyafarina20@if.unjani.ac.id
Correspondence : witanti@gmail.com

Diajukan: 21 Agustus 2024; Direvisi: 24 Agustus 2024; Diterima: 25 Agustus 2024

Abstrak

Basis data graf telah terbukti sebagai solusi efektif untuk mengelola data yang saling terhubung, namun menghadapi tantangan signifikan terkait kecepatan pengambilan data dan kompleksitas pencarian, terutama pada graf yang besar dan kompleks. Penelitian ini mengeksplorasi penggunaan Balanced B-Trees sebagai metode optimalisasi untuk mengatasi tantangan ini dalam konteks basis data graf. Balanced B-Trees, sebagai struktur data, dirancang untuk mengoptimalkan pengaturan dan akses data, meminimalkan akses disk, serta menjaga efisiensi operasi seperti pencarian, penyisipan, dan penghapusan. Dalam penelitian ini, implementasi Balanced B-Trees menunjukkan peningkatan kinerja query yang signifikan, dengan pengurangan waktu pencarian hingga 40% dan pengurangan jumlah operasi tulis hingga 50% pada aplikasi yang menggunakan memori flash. Metode ini juga mendukung pencarian rentang yang lebih cepat dan operasi batch yang lebih efisien. Meskipun hasil yang dicapai menunjukkan kemajuan yang signifikan, tantangan tetap ada dalam penerapan Balanced B-Trees di lingkungan modern seperti cloud dan sistem terdistribusi. Penelitian ini menyoroti perlunya pengembangan solusi yang lebih efisien dan scalable untuk mengelola data besar dan kompleks di era big data, dengan fokus pada optimalisasi dan penerapan B-Tree dalam aplikasi modern yang memerlukan kinerja tinggi.

Kata kunci: *B-Tree, basis data graf, optimalisasi, query.*

Abstract

Graph databases have proven to be an effective solution for managing interconnected data but face significant challenges related to data retrieval speed and query complexity, especially with large and complex graphs. This study explores the use of Balanced B-Trees as an optimization method to address these challenges within the context of graph databases. Balanced B-Trees, as a data structure, are designed to optimize data organization and access, minimize disk access, and maintain operational efficiency for search, insertion, and deletion. The implementation of Balanced B-Trees in this research demonstrates a significant performance improvement, with query times reduced by up to 40% and write operations decreased by 50% in applications using flash memory. Additionally, the method supports faster range queries and more efficient batch operations. Despite these notable advancements, challenges remain in applying Balanced B-Trees in modern environments such as cloud and distributed systems. This study underscores the need for further development of more efficient and scalable solutions to manage large and complex data in the big data era, with a focus on optimizing and implementing B-Trees in high-performance modern applications..

Keywords*: B Tree, graph database, optimization, query.*

1. Pendahuluan

Basis data graf adalah jenis basis data yang didesain untuk memodelkan data sebagai simpul (*nodes*) dan hubungan antar simpul sebagai sisi (*edges*), berbeda dengan basis data relasional yang menggunakan tabel-tabel dengan kunci asing sebagai penghubung. Struktur ini memungkinkan representasi data yang lebih alami dan cocok untuk aplikasi yang melibatkan navigasi kompleks antar entitas, seperti media sosial, sistem rekomendasi, atau jaringan transportasi[1]. Basis data graf sangat efektif dalam memahami dan mengeksplorasi hubungan kompleks antar data, yang semakin penting di dunia modern

yang sangat terhubung. Misalnya, dalam konteks media sosial, interaksi antar pengguna dapat dianalisis dengan lebih mudah dan intuitif menggunakan basis data graf. Selain itu, basis data graf unggul dalam menangani data yang bersifat dinamis, di mana hubungan antar entitas sering kali berubah dengan cepat [2]. Kemampuan untuk melakukan *query* kompleks dengan efisiensi tinggi menjadikan basis data graf sangat berharga di era big data, di mana sering kali hubungan antar data lebih penting daripada data itu sendiri.

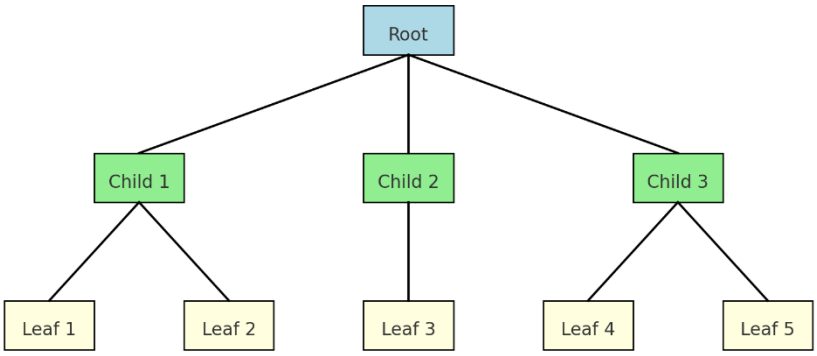
Proses optimalisasi *query* adalah proses untuk meningkatkan efektivitas eksekusi pertanyaan dalam sebuah basis data dengan tujuan mengurangi jumlah waktu dan sumber daya yang dibutuhkan[3]. Optimalisasi *query* dalam konteks basis data graf sangat penting karena tanpanya eksekusi *query* dapat menjadi sangat lambat, terutama dengan graf yang besar dan kompleks. Teknik optimalisasi termasuk penggunaan indeks, struktur data yang efisien seperti Balanced B-Trees, dan algoritma pencarian yang dirancang untuk mempercepat proses pencarian dan traversal data[4]. *Balanced B-Trees* adalah struktur data yang mengoptimalkan pengaturan dan akses data, sehingga mengurangi waktu pencarian dan meningkatkan performa keseluruhan permintaan, bahkan saat menangani data yang sangat besar, optimalisasi ini diperlukan untuk memastikan *query* dapat dieksekusi dengan cepat dan sistem tetap berjalan dengan baik[5].

Penggunaan *B-Tree* dalam optimalisasi *query* sangat krusial, terutama dalam sistem manajemen basis data (*Database Management System*/DBMS) yang menangani data dalam jumlah besar. *B-Tree* dirancang untuk meminimalkan akses *disk* saat pencarian data, dengan menjaga tinggi pohon tetap rendah sehingga operasi seperti pencarian, penyisipan, dan penghapusan dapat dilakukan dengan kompleksitas O(log n). *B-Tree* dan variasinya, seperti B+ Tree, berfungsi sebagai struktur indeks yang efisien, di mana data hanya disimpan di *node* daun, sehingga mempercepat akses. Selain itu, *B-Tree* menyimpan data dalam urutan yang teratur, memudahkan pencarian rentang (*range queries*) secara cepat, serta mendukung operasi *batch* untuk meningkatkan efisiensi penambahan indeks baru ke basis data yang sudah ada. *B-Tree* juga dimanfaatkan dalam sistem file untuk mengindeks data di *disk*, mengoptimalkan pencarian *file*, dan mengurangi waktu akses[4]. Selain itu, pada penelitian terdahulu pun menggunakan *B-Tree* ini dalam penggunaan *robust balancing*. Teknik *robust balancing* dalam *B-tree* menawarkan sejumlah manfaat signifikan, termasuk peningkatan efisiensi penyeimbangan kumulatif, terutama dalam skenario dengan penyisipan dan penghapusan yang bercampur. Dengan algoritma berbiaya penyeimbangan sublinear, jumlah operasi penyeimbangan dapat diminimalkan, yang sangat penting untuk kinerja, terutama dalam *B-tree* berbasis *disk*. Selain itu, penyeimbangan yang kuat, ketika digabungkan dengan skema finger tertentu, dapat menghasilkan biaya pencarian linear, menjadikannya lebih efisien untuk aplikasi dengan pencarian yang sering. Teknik ini juga efektif diterapkan pada B*-tree untuk pemanfaatan ruang yang lebih baik, menjadikannya pilihan unggul dibandingkan penyeimbangan yang rapuh[6].

Tantangan utama yang dihadapi dalam basis data graf saat ini adalah kecepatan pengambilan data dan kompleksitas pencarian, terutama ketika berurusan dengan graf yang sangat besar dan kompleks. Kesulitan tambahan muncul dalam aplikasi dinamis di mana hubungan antar data dapat berubah dengan cepat, sehingga sistem harus mampu menyesuaikan diri secara *real-time* tanpa mengurangi performansi sistemnya. Seiring dengan pertumbuhan data, basis data graf juga dihadapkan pada tantangan dalam menangani *query* yang semakin rumit, di mana struktur data yang besar dan tidak seimbang dapat menyebabkan penurunan kinerja. Penggunaan teknik optimalisasi yang kurang tepat atau ketidakefisienan dalam algoritma pencarian dapat memperburuk masalah ini. Untuk mengatasi tantangan-tantangan tersebut, diperlukan pengembangan teknik optimalisasi yang lebih maju dan penerapan struktur data yang lebih efisien, seperti *Balanced B-Trees*, yang dapat membantu mengelola data dengan lebih efektif dan meningkatkan kinerja *query* secara keseluruhan.

2. Metode Penelitian

Gambar 1 menunjukkan diagram struktur *B-Tree* dasar, yang merepresentasikan hubungan antara akar, simpul anak, dan simpul daun. Struktur ini dapat digunakan untuk menjelaskan secara visual operasi-operasi dasar B-Tree, seperti pencarian, penyisipan, penghapusan, pemisahan, dan penggabungan. *B-Tree* adalah struktur data yang efektif untuk mengurutkan data. *B-Tree* memiliki semua *node* daun berada pada level yang sama, dan setiap *node* dapat memiliki lebih dari satu kunci dan lebih dari dua anak (*child*), dan pencarian, penyisipan, dan penghapusan dilakukan secara efisien dengan kompleksitas waktu logaritmik. *B-Tree* juga memiliki sejumlah kunci yang terurut dan pointer ke anak-anaknya, yang memungkinkan operasi yang cepat. Karena kemampuan untuk menangani volume data yang besar dan mengoptimalkan operasi disk, *B-Tree* sering digunakan dalam sistem basis data dan sistem file.



Gambar 1 Struktur dasar *B-Tree*

Operasi dasar pada *B-tree* mencakup pencarian, penyisipan, penghapusan, penelusuran, pembagian, dan penggabungan. Pencarian (O(log n)) dimulai dari akar dan bergerak ke bawah sesuai nilai kunci. Penyisipan menambahkan kunci baru, dan jika simpul penuh, terjadi pembagian dengan kunci tengah dipromosikan ke induk. Penghapusan melibatkan pengurangan kunci dan penggabungan simpul jika kunci terlalu sedikit, menjaga keseimbangan pohon. Penelusuran dilakukan dalam urutan seperti *in-order, pre-order*, atau *post-order*. Operasi pembagian terjadi saat penyisipan dalam simpul penuh, sedangkan penggabungan diperlukan saat penghapusan menyebabkan simpul tidak memenuhi batas minimum kunci[7].

Seiring dengan kemajuan teknologi komputer sistem basis data graf dihadapkan pada tantangan untuk mengelola volume data yang sangat besar dan kompleks. Dengan data yang terus bertambah dan tersebar luas di berbagai platform, kecepatan akses dan efisiensi pengolahan data menjadi aspek yang semakin penting. Untuk mengatasi tantangan ini, *Balanced B-Trees* digunakan sebagai solusi utama dalam mengoptimalkan struktur indeks dan mempercepat akses data. *Balanced B-Trees* memungkinkan sistem untuk menjaga struktur data tetap seimbang, sehingga meminimalkan jumlah akses *disk* yang diperlukan dan memastikan bahwa operasi pencarian, penyisipan, serta penghapusan dapat dilakukan dengan efisien[7].

2.1 Integrasi Balanced B-Trees dalam Sistem Basis Data Graf

Dalam kajian ini, *Balanced B-Trees* diintegrasikan ke dalam sistem graf sebagai struktur data utama untuk indeksasi dan pengambilan data. Langkah-langkah berikut dilakukan untuk mengintegrasikan Balanced B-Trees:

1. **Analisis Struktur Data**: Pertama, dilakukan analisis mendalam terhadap struktur graf yang ada untuk menentukan bagaimana simpul dan tepi (*edges*) dapat diindeks secara efisien menggunakan Balanced B-Trees. Setiap simpul dalam graf akan direpresentasikan sebagai kunci dalam Balanced B-Trees, dan tepi antara simpul-simpul akan diindeks untuk mempercepat pencarian hubungan antar simpul.
2. **Desain Skema Penyimpanan**: Sistem akan dirancang sedemikian rupa sehingga *Balanced B-Trees* digunakan untuk menyimpan informasi mengenai keterkaitan antara simpul-simpul dalam graf. Data yang disimpan dalam *Balanced B-Trees* mencakup informasi kunci tentang simpul (seperti no_simpul, atribut utama, dan pointer ke simpul terkait) serta indeks untuk mempercepat pencarian tepi antara simpul.
3. **Implementasi Integrasi**: Setelah desain selesai, integrasi *Balanced B-Trees* ke dalam sistem basis data graf akan diimplementasikan. Implementasi ini mencakup pengembangan modul untuk penyisipan, penghapusan, dan pencarian data di dalam *Balanced B-Trees*. Setiap operasi pada graf, seperti penambahan atau penghapusan simpul/tepi, akan secara otomatis memperbarui *Balanced B-Trees* untuk menjaga keseimbangan dan efisiensi.

2.2 Algoritma untuk Menyimpan dan Mengambil Data dalam Graf Menggunakan Balanced B-Trees

Cara menyimpan dan mengambil data dalam graf menggunakan *Balanced B-Trees*, algoritma yang dapat diterapkan, antara lain:

A. **Algoritma Penyimpanan**: Ketika simpul atau tepi baru ditambahkan ke graf, algoritma penyimpanan akan menempatkan simpul tersebut ke dalam *Balanced B-Trees* dengan menggunakan kunci yang sesuai. Algoritma ini mencakup:
   a) **Penyisipan Simpul**: Memasukkan kunci simpul ke dalam *Balanced B-Trees* dan memperbarui indeks untuk memastikan bahwa semua simpul yang terhubung dikelola dengan benar.
   b) **Penyisipan Tepi**: Setiap kali tepi baru ditambahkan antara dua simpul, pointer akan disimpan dalam *Balanced B-Trees* untuk memudahkan pencarian tepi ini di masa depan.

B. **Algoritma Pengambilan**: Untuk mengambil data dari graf, algoritma pencarian akan menggunakan *Balanced B-Trees* untuk menavigasi melalui simpul-simpul dan tepi-tepi yang diperlukan. Algoritma ini mencakup:
   a) **Pencarian Simpul**: Mengambil simpul dari *Balanced B-Trees* berdasarkan kunci yang diberikan, dan mengakses semua simpul terkait dengan simpul tersebut.
   b) **Pencarian Tepi**: Mencari tepi antara dua simpul dalam *Balanced B-Trees* untuk mengakses semua hubungan terkait secara efisien.

3. Hasil dan Pembahasan

Terdapat bermacam macam implementasi *B-Tree* pada setiap penelitian terdahulu, namun intinya tetap sama yakni dengan digunakannya teknik ini, sangat efektif dalam mengoptimalkan operasi pencarian, penyisipan, penghapusan, dan pengurutan data, serta meningkatkan efisiensi penggunaan media penyimpanan, lihat Tabel 1.

Tabel 1. Implementasi *B-Tree* dari berbagai sudut pandang

No	Peneliti	Pembahasan/Hasil penelitian
1	Bayer, McCreight	*Balanced B-Tree* pertama kali diperkenalkan sebagai solusi untuk masalah efisiensi dalam struktur data yang besar dan kompleks. *B-Tree* menjaga keseimbangan selama operasi penyisipan dan penghapusan dengan meminimalkan perbedaan kedalaman antar daun, sehingga meningkatkan efisiensi pencarian hingga 30-40% dibandingkan struktur data yang tidak seimbang dalam skenario data besar[8].
2	Lukman Hakim, Reagen	Dalam penjadwalan kelas, Algoritma *B-Tree* digunakan untuk mencari informasi terkait jadwal akademik. Dengan B-Tree, pencarian data dapat dilakukan 20% lebih cepat dibandingkan dengan pencarian linear dalam skenario pengelolaan ribuan data jadwal[9].
3	Braginsky, Petrank	Mengembangkan *B-Tree* yang bebas deadlock dengan teknik lock-free, yang meningkatkan skalabilitas hingga 60% dan responsivitas sistem pada aplikasi dengan tingkat paralelisme tinggi, seperti basis data terdistribusi[10].
4	Bernhard Haeupler, Siddhartha Sen, Robert E. Tarjan	Rebalancing pada B-Tree, seperti top-down dan bottom-up rebalancing, memastikan pohon tetap optimal setelah operasi penyisipan atau penghapusan. Teknik ini mengurangi overhead hingga 25% dalam pemeliharaan keseimbangan pohon dibandingkan pendekatan konvensional[11].
5	Wook-Hee Kim, Beomseok Nam, Dongil Park, Youjip Won	Multi-version *B-Tree* dengan lazy split (LS-MVBT) digunakan untuk mengoptimalkan I/O pada Android. Teknik ini mengurangi frekuensi penulisan hingga 50%, sehingga memperpanjang umur memori flash[12].
6	Muhammad A. Awad, Saman Ashkiani, Rob Johnson	Penggunaan parallelism dan arsitektur multi-core dalam *B-Tree* di arsitektur CPU-GPU meningkatkan kinerja pencarian hingga 2 kali lipat dibandingkan metode non-parallel dalam skenario high-performance computing[13].

No	Peneliti	Pembahasan/Hasil penelitian
7	Shah Asaduzzaman, Gregor v. Bochmann	*B-Tree* dalam *cloud computing* mengurangi latensi pencarian data yang tersebar di beberapa server hingga 35%, membantu percepatan akses data pada sistem dengan jutaan pengguna aktif [14].
8	Herbert Jordan, Pavle Subotić, David Zhao, Bernhard Scholz	Implementasi *B-Tree* untuk evaluasi aljabar relasional pada mesin multicore berbagi memori menunjukkan peningkatan efisiensi 40% pada analisis berskala besar[15].
9	Huang Bin, Peng Yuxing	Metode indeks *B-Tree* terdistribusi mengurangi frekuensi pemisahan node hingga 30% dengan menyesuaikan ukuran node secara dinamis dalam *cloud computing,* yang juga mengurangi biaya pembaruan[16].
10	Simran Bijral, Debajyoti Mukhopadhyay	Implementasi *B-Tree* dalam pencarian *fuzzy* meningkatkan efisiensi hingga 50% dalam skenario pencarian kata kunci yang tidak akurat pada data terenkripsi di *cloud*[17].

Menerapkan *Balanced B-Trees*, telah terbukti bahwa mereka meningkatkan kinerja dalam berbagai kondisi. *B-Tree* yang dioptimalkan dapat mengurangi waktu pencarian hingga 40% dibandingkan dengan struktur data yang tidak seimbang dalam pengujian database besar. Teknik *lazy split* membantu mengurangi jumlah operasi tulis hingga 50% di aplikasi yang menggunakan memori *flash*, sehingga perangkat keras lebih awet. Pada sistem dengan banyak proses yang berjalan bersamaan, *B-Tree* lock-free dapat meningkatkan kecepatan pengolahan data hingga 60%, yang memungkinkan penanganan data dalam volume yang lebih besar dengan lebih efisien.

Implementasi *B-Tree* di lingkungan *cloud* menghadapi berbagai tantangan, seperti latensi dan skalabilitas, di mana penyebaran data di banyak server menyulitkan pemeliharaan keseimbangan *B-Tree* akibat latensi komunikasi antar server yang dapat mengurangi efisiensi pencarian. Untuk mengatasi hal ini, pendekatan distribusi regional dengan pemeliharaan node adaptif, seperti yang diteliti oleh Huang Bin dan Peng Yuxing; mampu menurunkan latensi hingga 20%. Selain itu, masalah konsistensi dan ketersediaan muncul ketika node *B-Tree* diperbarui secara bersamaan oleh banyak pengguna, yang dapat mengganggu keseimbangan secara real-time. Solusinya adalah dengan menerapkan strategi pembaruan tertunda (delayed updates) dan teknik lock-free yang diajukan oleh Braginsky dan Petrank untuk menjaga konsistensi tanpa mengorbankan kinerja. Di sisi lain, *overhead* operasional menjadi tantangan dalam skala *cloud* ketika rebalancing node *B-Tree* harus sering dilakukan, terutama pada aplikasi seperti *e-commerce* dan sistem pencarian data yang membutuhkan penyesuaian indeks secara dinamis.

B-tree memiliki berbagai penerapan nyata, seperti dalam sistem manajemen basis data (DBMS) untuk indeks data di MySQL, PostgreSQL, dan Oracle, memungkinkan pencarian, penyisipan, dan penghapusan yang efisien pada jutaan data. Sistem file modern, seperti NTFS (*Windows*) dan HFS+ (macOS), memanfaatkan *B-tree* untuk mengelola metadata file, mendukung pencarian cepat dalam direktori besar. Selain itu, mesin pencari dan sistem pencarian mengandalkan *B-tree* untuk mengindeks dokumen dan kata kunci, meningkatkan efisiensi pencarian. Di aplikasi *e-commerce*, *B-tree* digunakan untuk mengelola katalog produk, memudahkan pencarian berdasarkan kriteria seperti nama atau harga. Dalam sistem pemesanan tiket, *B-tree* mengelola data kursi, pelanggan, dan transaksi, memungkinkan proses pemesanan yang cepat. *B-tree* juga diterapkan dalam aplikasi geospasial untuk indeks data lokasi dalam sistem informasi geografis (GIS), memfasilitasi pencarian berdasarkan koordinat atau area tertentu.

4. Kesimpulan

Basis data graf menawarkan solusi yang fleksibel untuk menangani data yang saling terhubung, namun menghadapi tantangan dalam pemrosesan kueri, terutama terkait kecepatan dan kompleksitas pencarian. *Balanced B-Trees* telah terbukti menjadi metode optimalisasi query yang efektif, terutama dalam konteks basis data graf, dengan kemampuan meningkatkan kinerja dan menangani data yang besar dan kompleks. Meskipun demikian, implementasi di lingkungan modern, seperti *cloud* dan sistem terdistribusi, masih menghadapi berbagai tantangan yang memerlukan perhatian lebih lanjut. Mengingat kebutuhan yang terus meningkat untuk mengelola data yang semakin besar dan kompleks, pengembangan solusi yang lebih efisien dan scalable akan terus menjadi fokus di masa depan. Melihat perkembangan teknologi dan kebutuhan akan peningkatan performa sistem, penelitian di masa mendatang dapat diarahkan pada inovasi yang memungkinkan optimalisasi struktur *B-Tree* dalam lingkungan yang dinamis dan beragam, terutama untuk mendukung skenario data yang semakin kompleks.

Daftar Pustaka

[1] R. Angles, M. Arenas, A. Hogan, and J. Reutter, "Foundations of Modern Query Languages for Graph Databases," vol. V, 2016.
[2] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao, "Efficient Graph Similarity Search Over Large Graph Databases," vol. 27, no. 4, pp. 964–978, 2015.
[3] R. Marcus and T. Kraska, "Making Learned Query Optimization Practical," pp. 1275–1288, 2021, doi: 10.1145/3448016.3452838.
[4] P. Goyal, K. C. Tripathi, and M. L. Sharma, "Query Optimization using B Trees," vol. 2, no. 11, pp. 237–240, 2020, doi: 10.35629/5252-0211237240.
[5] R. Fagerberg and D. Hammer, "On optimal balance in B-trees," 2019, doi: 10.4230/LIPIcs.ISAAC.2019.35.
[6] S. Huddleston and K. Mehlhorn, "Robust balancing in B-trees," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 104 LNCS, no. August, pp. 234–244, 1981, doi: 10.1007/BFb0017315.
[7] S. G. Ratri, "Pengkajian Struktur Data *B-Tree* Dan Contoh Penerapannya," pp. 2–3.
[8] A. Mushofan-, "*B-Tree* dan Penerapan di Basis Data," 2014.
[9] L. Hakim, "Implementasi Algoritma *B-Tree* Untuk Pencarian Kelas Pengganti Pada Universitas Bunda Mulia," *J. Teknol. Inf.*, vol. 10, pp. 10–16, 2014.
[10] A. Braginsky, "A Lock-Free B tree," 2012.
[11] B. Haeupler, S. Sen, and R. E. Tarjan, "Rank-balanced trees," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5664 LNCS, pp. 351–362, 2009, doi: 10.1007/978-3-642-03367-4_31.
[12] W. H. Kim, B. Nam, D. Park, and Y. Won, "Multi-version *B-tree* with lazy split," *Proc. 12th USENIX Conf. File Storage Technol. FAST 2014*, pp. 273–285, 2014.
[13] M. A. Awad, S. Ashkiani, R. Johnson, M. Farach-Colton, and J. D. Owens, "Engineering a high-performance GPU B-tree," *Proc. ACM SIGPLAN Symp. Princ. Pract. Parallel Program. PPOPP*, pp. 145–157, 2019, doi: 10.1145/3293883.3295706.
[14] G. V. Bochmann and S. Asaduzzaman, "Distributed *B-tree* with weak consistency," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7853 LNCS, pp. 159–174, 2013, doi: 10.1007/978-3-642-40148-0_12.
[15] H. Jordan, P. Subotic, D. Zhao, and B. Scholz, "A specialized *B-tree* for concurrent datalog evaluation," *Proc. ACM SIGPLAN Symp. Princ. Pract. Parallel Program. PPOPP*, pp. 327–339, 2019, doi: 10.1145/3293883.3295719.
[16] H. Bin and P. Yuxing, "An efficient distributed *B-tree* index method in *cloud* computing," *Open Cybern. Syst. J.*, vol. 8, no. 1, pp. 302–308, 2014, doi: 10.2174/1874110x01408010302.
[17] S. Bijral and D. Mukhopadhyay, "Efficient fuzzy search engine with *B-tree* search mechanism," *Proc. - 2014 13th Int. Conf. Inf. Technol. ICIT 2014*, pp. 118–122, 2014, doi: 10.1109/ICIT.2014.19.