HIERARCHICAL BLOOM FILTER UNTUK EFISIENSI PENYIMPANAN DATA AKUN *(USERNAME DAN PASSWORD)*

Muh Fauzy Putra Bojo[1], Gia S Wulandari[2], Farah Afianti[3]

[1,2,3] Universitas Telkom, Bandung

[1]fauzyputra@students.telkomuniversity.ac.id, [2]giaseptiana@telkomuniversity.ac.id, [3]farahafi@telkomuniversity.ac.id

Abstrak

***Password* dan *username* merupakan informasi penting dan sensitif dari pengguna. Sebagian besar *database* pengguna, menyimpan *username* dan *password.* Maka dari itu, dibutuhkan metode penyimpanan informasi yang membutuhkan lebih sedikit penyimpanan, pemrosesan kueri yang lebih cepat, dan melindungi informasi yang telah disimpan. Salah satu metode penyimpanan akun dan password saat ini adalah dengan menggunakan fungsi *hash SHA-1*. Fungsi *Hash SHA-1* adalah algoritma yang berfungsi untuk mengubah teks menjadi sederetan karakter acak yang memiliki jumlah panjang karakter yang sama. Namun, salah satu kekurangan dari *hash SHA-1* adalah semakin meningkatnya jumlah pengguna maka penggunaan penyimpanan juga akan semakin besar. Maka dari itu, dibutuhkan Hierarchical Bloom Filter untuk dapat melakukan efisiensi terhadap penyimpanan data akun sehingga tidak terjadi peningkatan terhadap penyimpanan data akun apabila jumlah pengguna juga meningkat. Untuk mengatasi masalah ini, penulis mengusulkan kerangka kerja berbasis *Hierarchical Bloom Filter* (HBF). HBF telah diperkenalkan untuk mengatasi masalah pencocokan *sub-string*. Pada penelitian ini, penulis melakukan pengamatan terhadap 10.000 data yang diinputkan ke dalam HBF dengan program yang dibangun dengan bahasa pemrograman *python*. Setelah diimplementasikan, penulis mendapatkan hasil pemapatan data di atas 70% dengan waktu *insert* dan *query* yang linier terhadap ukuran data.**

Kata kunci : hierarchical bloom filter, pemapatan data, penyimpanan password, peyimpanan akun

Abstract

Password and username are important and sensitive information of users. Most user databases, store usernames and passwords. Therefore, there is a need for information storage methods that require less storage, faster query processing, and protect the information that has been stored. One of the current methods of storing accounts and passwords is to use the SHA-1 hash function. Which is an algorithm that functions to convert text into a series of random characters that have the same number of characters in length. However, one of the drawbacks of the SHA-1 hash is that as the number of users increases, the storage usage will also increase. Therefore, a Hierarchical Bloom Filter is needed to be able to perform efficiency on account data storage so that there is no increase in account data storage if the number of users also increases. To overcome this problem, the author proposes a Hierarchical Bloom Filter (HBF) based framework. HBF has been introduced to address the problem of sub-string matching. In this study, the authors observed 10,000 data entered into the HBF with a program built with the python programming language. Once implemented, the authors get the results of data compression above 70% with insert and query times that are linear to the size of the data.

Keywords: : hierarchical bloom filter, data compression, password storage, account storage

1. Pendahuluan

Pada saat ini pertumbuhan teknologi informasi telah berpengaruh pada hampir semua aspek kehidupan manusia. Internet tidak lagi mempedulikan batas geografis suatu negara, sehingga bukan lagi sebuah kesulitan bagi seseorang untuk berkomunikasi dari jarak jauh, mengirim dan mencari informasi. Seluruh kegiatan tersebut dapat dilakukan dengan cepat, efisien, dan terhitung murah melalui internet. Namun tidak bisa dipungkiri, fakta membuktikan bahwa teknologi informasi membutuhkan ruang penyimpanan data yang besar untuk menyimpan berbagai data pengguna. Sehingga, teknologi informasi atau aplikasi dapat digunakan dengan benar dan nyaman oleh pengguna. Semua data pengguna disimpan dalam database yang terhubung dengan aplikasi yang digunakan oleh pengguna. Ada beberapa metode otentikasi berdasarkan kebijakan administrator sistem dan infrastruktur yang tersedia, salah satunya adalah penggunaan *password* dan *username* [1]. *Password* dan *username* adalah informasi penting dan sensitif bagi pengguna. Sebagian besar *database* pengguna menyimpan *username* dan *password*. Maka

dari itu, dibutuhkan metode penyimpanan informasi yang membutuhkan lebih sedikit penyimpanan, pemrosesan kueri yang lebih cepat, dan melindungi informasi yang telah disimpan.

Salah satu metode penyimpanan *username* dan *password* saat ini adalah dengan menggunakan fungsi *hash SHA-1(Secure Hash Algoritm-1).* Fungsi *hash SHA-1* adalah algoritma yang berfungsi untuk mengubah teks menjadi sederetan karakter acak yang memiliki jumlah panjang karakter yang sama [2]. *SHA-1* merupakan salah satu algoritma kriptografi yang aman karena proses penghitungan *SHA-1* dilakukan secara *invisible* untuk mencari *string* yang sesuai agar dapat menghasilkan *message digest* yang sama. Setiap perubahan yang terjadi selama proses pengiriman pesan akan menghasilkan *message digest* yang berbeda. *Message digest* sendiri merupakan keluaran maupun masukan dari *SHA-1* dengan panjang jarak berkisar 160 sampai 512 bit tergantung algoritmanya. Namun, salah satu kelemahan dari fungsi hash SHA-1 adalah seiring bertambahnya jumlah pengguna, penggunaan penyimpanan juga akan meningkat.

Oleh karena itu, perlu dilakukan tindakan untuk memperkecil ukuran ruang penyimpanan, salah satunya dengan melakukan kompresi pada media penyimpanan data pengguna. Untuk kompresi data teks/string, metode yang umum digunakan adalah metode *deflate* (Philips Kats). Metode ini merupakan kombinasi dari algoritma LZ77 dan algoritma *Huffman*. Cara kerjanya adalah dengan menghilangkan *double string*. Metode *deflate* diuji pada tahun 2017, dan hasil dari metode ini adalah: mengompresi data asli 184-bit menjadi data 43-bit [3]. Namun, metode ini kurang cocok digunakan untuk *username* dan *password*. Untuk *username* dan *password*, kita tidak bisa menghilangkan karakter, karena akan menyebabkan *error* ketika program meminta untuk melakukan verifikasi ulang *username* dan *password*. Oleh karena itu, penulis memberikan metode kompresi data menggunakan *Bloom Filter* (BF). BF menawarkan jumlah ruang penyimpanan yang lebih kecil dibandingkan dengan penyimpanan pada umumnya. Selain itu, karena BF hanya membutuhkan ruang penyimpanan yang relatif kecil, sehingga pemrosesan kueri juga akan berjalan lebih cepat. Pada tugas akhir ini, penulis mengamati hasil proses penyimpanan *username* dan *password* dengan menggunakan HBF yang merupakan modifikasi dari BF, dilihat dari ruang penyimpanan dan waktu kompresi yang diperlukan. Adapun data *username* dan *password* yang digunakan pada tugas akhir ini dibangkitkan secara acak sejumlah 10.000 pasang *username* dan *password*. Dengan panjang *username* dan *password* yang fixed, yaitu 10 karakter *username* dan 8 karakter *password*. Dari data tersebut dilakukan analisis penurunan ukuran penyimpanan dan waktu pemrosesan kueri.

Bagian 2 pada Tugas Akhir ini menggambarkan secara umum tentang HBF dan pemapatan data secara umum. Untuk bagian 3, disajikan rancangan dari sistem yang penulis buat, sedangkan untuk bagian 4 memberikan rincian tentang hasil dan analisis sistem. Pada bagian 5 penulis menjelaskan kesimpulan dari penelitian ini.

2. Studi Terkait

2.1 Pemapatan Data

Saat ini kebutuhan akan kapasitas ruang penyimpanan yang besar menjadi semakin penting. Kebutuhan ini terjadi akibat data yang harus disimpan dari waktu-kewaktu semakin bertambah banyak, sehingga kapasitas penyimpanan yang dibutuhkan juga semakin bertambah. Salah satu solusi untuk masalah ruang penyimpanan adalah dengan memampatkan *(compress)* data yang akan disimpan. Pada dasarnya teknik pemampatan data ini adalah dengan menjadikan ukuran file lebih kecil dengan melakukan pengkodean baru terhadap isi file [4].

Pemampatan data yang dilakukan dapat dikategorikan menjadi 2, *lossless* dan *lossy*. *Lossless* sendiri adalah teknik pemampatan data namun tidak menghilangkan informasi asli data tersebut. Kategori ini banyak digunakan untuk pemampatan data berupa teks. Sedangkan kategori *lossy* adalah teknik memampatkan data dengan membuang sedikit informasi asli data, hal ini dilakukan karena data tersebut biasanya menumpuk nilai yang sebenarnya tidak dibutuhkan atau nilai yang terduplikat. Contoh pemampatan informasi data yang digunakan pada kategori ini adalah gambar atau suara.

2.2 Algoritma SHA-1

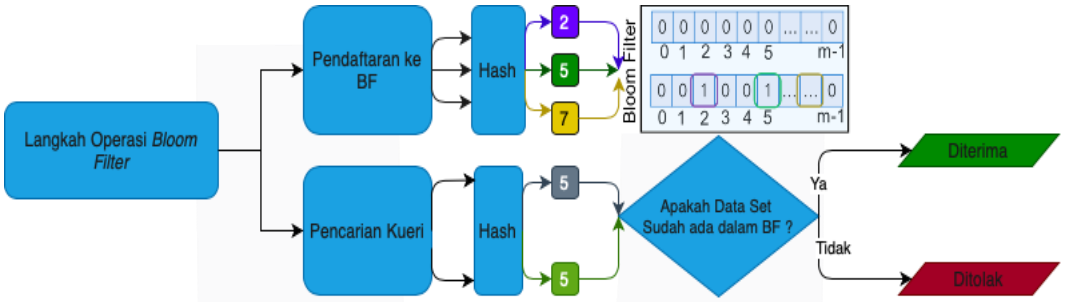
*Secure Hash Algoritm (*SHA) adalah sebuah fungsi *hash* satu arah yang diciptakan oleh NIST dan digunakan bersama DSS (*Digital Signature Standard*). *SHA* dianggap sebagai penerus dari pendahulunya yaitu, MD5. *SHA* disebut sangat aman dalam penggunaannya karena dirancang sedemikian rupa sehingga didalam komputasi tidak mungkin menemukan pesan yang berkoresponden dengan *message digest* yang diberikan. Fungsi *SHA* yang paling sering dan umum digunakan yaitu *SHA-1*. Fungsi ini telah diimplementasikan ke dalam berbagai aplikasi dan prokotol keamanan seperti SSL, TLS, SSH, PGP, Ipsec dan S/MIME. Pada tahun 1993, anggota pertama keluarga *SHA* yang dipublikasikan yaitu *SHA*-0 yang sering

diacu sebagai *SHA* saja. *SHA*-1 kemudian baru dipublikasikan pada tahun 1995. Selain *SHA-0* dan *SHA-1*, ada beberapa varian *SHA* lain yang telah dipublikasika, yaitu *SHA*-224, *SHA*-256, *SHA*-384, dan *SHA*-512.

Fungsi *hash SHA-1* adalah algoritma yang berfungsi untuk mengubah teks menjadi sederetan karakter acak yang memiliki jumlah panjang karakter yang sama [2]. Fungsi ini merupakan salah satu algoritma kriptografi yang aman karena proses penghitungannya yang dilakukan secara *invisible* untuk mencari *string* yang sesuai agar dapat menghasilkan *message digest* yang sama. Setiap perubahan yang terjadi selama proses pengiriman pesan akan menghasilkan *message digest* yang berbeda. *Message digest* sendiri merupakan keluaran maupun masukan dari *SHA-1* dengan panjang jarak berkisar 160 sampai 512 bit tergantung algoritmanya. Namun, salah satu kelemahan dari fungsi hash SHA-1 adalah seiring bertambahnya jumlah pengguna, penggunaan penyimpanan juga akan meningkat .

2.3 Hierarchical Bloom Filter

Hierarchical Bloom Filter merupakan perbaikan dari *Bloom Filter*. *Bloom filter* (BF) sendiri merupakan struktur data probabilistik hemat ruang yang digunakan untuk memperkirakan penyaringan keanggotaan dan secara efektif dapat meningkatkan strategi suatu organisasi dalam hal sumber daya [5]. BF memungkinkan untuk merancang sebuah *database* yang membutuhkan lebih sedikit ruang penyimpanan dibandingkan dengan yang asli. Karena ukuran yang lebih kecil, sehingga pemrosesan kueri berjalan lebih cepat. Kemudian, pola keamanan akan tercapai karena selama perancangan BF, *hashing* satu arah dilakukan pada pola yang belum dilatih. Selain itu, BF juga diterapkan dalam beragam sistem Jaringan untuk meningkatkan kinerja sistem dan mengurangi konsumsi memori, misalnya, *Named-Data Networking, Software-Defined Networks, and Wireless Sensor Networking*.



Gambar 1. Langkah pendaftaran elemen dan query pencarian anggota pada *bloom filter*

Prinsip kerja pada BF meliputi dua proses, yaitu : pendaftaran elemen dan *query processing* untuk menguji keanggotaan. Beberapa parameter yang terkait dengan BF adalah : jumlah elemen yang akan dimasukkan ke dalam BF (n), tingkat *False Positive* (FP), jumlah fungsi *hash* (k), dan ukuran dari BF itu sendiri (m). Untuk sejumlah n elemen dan probabilitas dari FP (*false positive,)* (dimana FP [0, 1]), ukuran dari BF (m) dan jumlah optimal dari fungsi *hash* (k) dapat ditentukan untuk mempertahankan tingkat FP yang diinginkan. Keluaran dari fungsi *hash* adalah menunjukkan posisi atau indeks pada BF, dimana template harus dimasukkan. Karena keluaran fungsi *hash* dapat melebihi m, fungsi modulus diterapkan pada keluaran untuk mendapatkan lokasi yang dibatasi dalam rentang [0, m]. Lalu setelah itu, bit yang disebutkan diatas diatur ke "1" untuk menyelesaikan penyisipan template.

Pada tahun 2004, Yifeng Zhu dkk meggunakan modifikasi BF dengan memberikan peningkatan dalam proses *insert* dan *query*nya yang dikenal dengan *Hierarchical Bloom Filter* (HBF) [6]. Kemudian pada tahun 2019, Sumaiya Shoumaji dkk juga menggunakan HBF untuk melakukan penelitian dengan studi kasus pada sistem biometrik. Dalam penelitian tersebut shoumaji mendapatkan 92,05% reduksi pada penyimpanan, dan 99,82% rata-rata reduksi pada waktu kueri *Hierarchical bloom filter* (HBF) digunakan untuk mendukung pencocokan *substring* dan meningkatkan akurasi pencocokan [7]. Struktur HBF menyediakan alat untuk memeriksa bagian dari *string*. Operasi dasar HBF adalah membagi inputan *string* ke beberapa blok dengan ukuran tetap. BF standar digunakan untuk menyisipkan blok yang terpisah dan memberikan granularitas ukuran blok yang memeriksa substring. Ketika melakukan pencocokan *substring*, bisa terjadi pelaporan *False Positive*. Dalam struktur ini, kumpulan BF standar digunakan untuk meningkatkan ukuran blok. Ketika string baru ditambahkan ke dalam struktur, pertama kali dibagi menjadi blok dan kemudian mulai dari tingkat

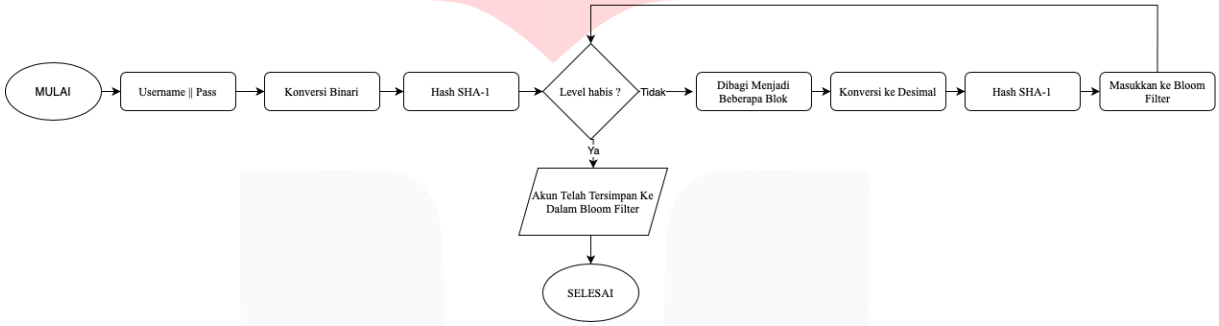
hierarki terendah, blok ditambahkan ke dalam filter. Pada tingkat berikutnya, penggabungan pertama dari dua blok berturut-turut dilakukan dan kemudian ditambahkan di tingkat berikutnya *(yaitu,*tingkat kedua) hierarki [8]. Proses yang sama dilanjutkan untuk tingkat hierarki yang tersisa. HBF terdiri dari BF standar dengan FP yang ditentukan dalam program. Template pada HBF dibagi menjadi beberapa blok. Blok terkecil dimasukkan ke dalam BF yang berlokasi pada level terendah. Kemudian, semua *string* dimasukkan kedalam BF pada level tertinggi. Tujuan dari HBF adalah untuk menentukan batas tertinggi pada jumlah BF.

HBF terdiri dari BF standar dengan FP yang ditentukan pada program. Template pada HBF dibagi menjadi beberapa blok. Blok terkecil dimasukkan ke dalam BF yang berlokasi pada level terendah. Lalu, semua *string* dimasukkan kedalam BF pada level tertinggi. Tujuan dari HBF adalah untuk menentukan batas tertinggi pada jumlah BF.

3. Sistem yang Dibangun

Pada HBF terdapat 2 proses utama yaitu, *entry* dan *query*. Proses *entry* bertujuan untuk mendaftarkan keanggotaan pada HBF, sedangkan proses *query* bertujuan untuk mengecek keanggotaan pada data set yang telah didaftarkan tadi. Kedua proses ini akan dijelaskan secara rinci pada bagian dibawah ini.

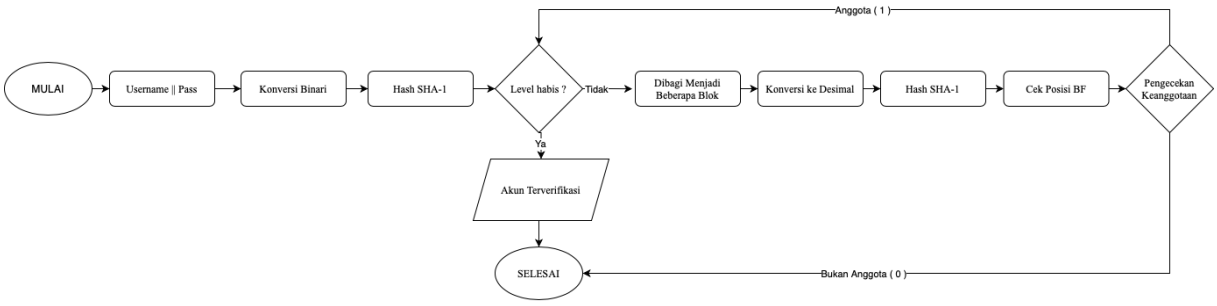
3.1 Entry Hierarchical Bloom Filter



Gambar 2 Proses *entry hierarchical bloom filter*

Pada Gambar 2 , informasi yang diinputkan adalah *username* dan *password.* Informasi ini digabungkan (*concatenate)* menjadi satu inputan *string* yang kemudian diubah menjadi *binary*. Kemudian dilakukan *hashing* dengan fungsi *SHA- 1,* lalu dilanjutkan ke proses selanjutnya, yaitu mengecek semua level telah habis atau belum. Jika level telah habis, maka proses *entry* selesai. Namun, jika level belum habis maka langkah selanjutnya adalah membagi informasi yang telah diubah tadi dibagi sesuai dengan nilai *block*. Langkah selanjutnya adalah konversi nilai tadi menjadi desimal, kemudian dilakukan *hash*. Setelah *hashing*, nilai tersebut dimasukkan ke dalam BF. Setelah dimasukkan pada BF, sistem kembali ke proses pengecekan level, sampai semua level telah habis.

3.2 Cek *Membership*



Gambar 3 Proses pengecekan keanggotan HBF

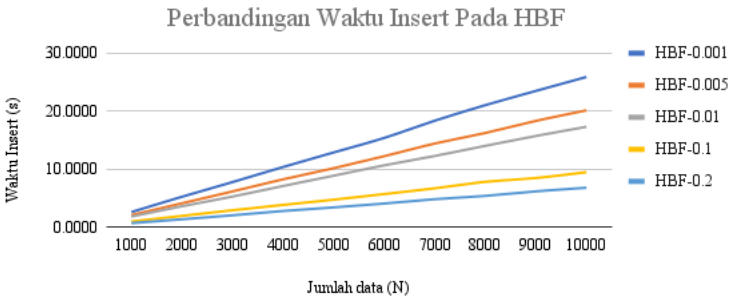
Proses pada Gambar 3 ini bertujuan untuk memeriksa keanggotaan akun pada HBF, nilai 0 jika bukan anggota dan 1 jika anggota. Proses pengecekan dimulai dengan memasukkan informasi *username* dan *password* dan kemudian diubah menjadi *binary*. Langkah selanjutnya hampir sama dengan langkah-langkah yang dilakukan pada proses *entry*. Langkah yang membedakan adalah setelah nilai di *hash* sistem akan melakukan pengecekan keanggotaan, jika nilai 0 maka bukan anggota dan proses selesai. Namun, jika nilai 1 proses akan kembali ke langkah pengecekan apakah level sudah habis atau belum. Jika level sudah habis maka informasi yang dimasukkan tadi telah diverifikasi sebagai anggota.

4 Evaluasi

Dalam bagian ini, dibahas 3 proses utama yaitu proses menghitung waktu *insert*, proses waktu kueri dan proses pemapatan penyimpanan dengan menggunakan HBF. Data yang digunakan dalam penelitian ini berjumlah 10.000 data yang di*generate* secara random, yang terdiri dari *username* dengan panjang string 10 dan yang terakhir *password* dengan panjang string 8. Perhitungan waktu *insert* dan *query*, serta besar pemapatan dilakukan pada nilai FP yang beragam, yaitu, FP 0,001, 0,005, 0,01, 0,1, dan 0,2. Gambar 4 menunjukkan perbandingan waktu *insert* pada HBF. HBF dengan nilai *false positve* 0,2 menghasilkan waktu *insert* yang lebih cepat dibandingkan dengan nilai *false positive* 0,001.

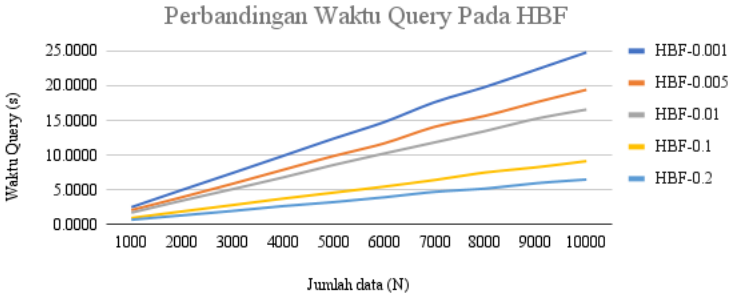
Tabel 1. Contoh data *username* dan *password*

Username	Password
0O32FYJ87Y	1GPYNSTN
P6DTA6CE46	M1FQF6UI
OZ4KKTXR4H	YLYXRI18
CPL6FCDRWB	OVP0D5NQ
1A3GVHD04N	XFEMKNM5
W43QH5HEVP	PUQ648MJ
YHHBS0XSKT	485HCNKP
SL98FHY7FB	X432ML53
JTBK2ZLD3N	X0QQVF7L



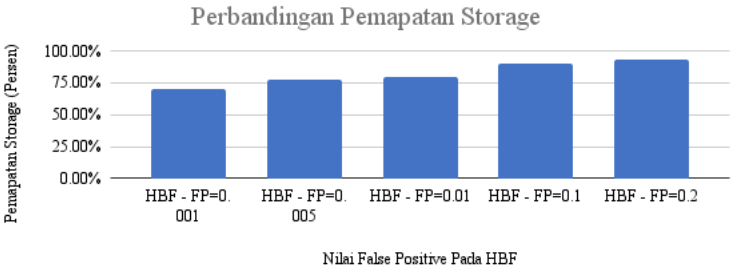
Gambar 4 Perbandingan Waktu Insert pada HBF

Kemudian pada Gambar 5 , HBF dengan nilai 0,2 juga menghasilkan waktu *query* yang lebih cepat dibandingkan dengan 0,001. Terlihat bahwa semakin kecil nilai FP yand ditetapkan, semakin besar waktu *query* yang dibutuhkan. Namun, dengan nilai FP 0,001, waktu *query* untuk data yang berisi 10.000 *username* dan *password* masih kurang dari 25 detik, dan dapat kita lihat pada Gambar 5 bahwa kompleksitas waktu *query* ini linier terhadap besarnya data yang kita miliki.



Gambar 5 Perbandingan Waktu Query pada HBF

Pada proses terakhir di Gambar 6 dari 3 proses utama yang dijelaskan pada bab ini, pemapatan storage HBF dengan nilai *false positive* 0,2 berhasil melakukan pemapatan *storage* sebesar 91,63% dan nilai pemapatan *storage* terkecil dihasilkan dari nilai *false positive* 0,001 dengan persentase sebesar 70,05%. Dari hasil pengujian sistem terlihat bahwa nilai FP >= 0.001 dapat memampatkan data lebih dari 70%.



Gambar 6 Perbandingan Pemapatan Storage

Meskipun nilai *false positive* 0,2 menghasilkan waktu insert, waktu query, dan pemapatan storage yang paling baik, penulis menilai hasil ini kurang bagus karena akan memberikan 20% hasil *false positive.* Hal ini dinilai kurang bagus karena bisa terjadi suatu kondisi salah pengenalan keanggotaan, dimana *username* dan *password* yang teridentifikasi anggota namun tidak pernah ada dalam dataset. Ketika kita menggunakan nilai *false positive* 0,001, hasil dari waktu dan pemapatan storagenya masih terhitung relatif bagus, sehingga HBF masih dapat digunakan untuk memampatkan data *username* dan *password*.

5 Kesimpulan

Berdasarkan analisis penulis terhadap sistem yang dibangun dan mengacu pada hasil pengamatan selama proses pengujian, maka penulis dapat menyimpulkan sebagai berikut :

1) HBF dapat digunakan untuk memperkecil ruang penyimpanan *username* dan *password*.
2) Proses *insert* dan *query* pada HBF memiliki kompleksitas waktu yang linier terhadap ukuran dataset. Pada data yang berisi 10,000 *username* dan *password* , dengan FP 0,001, dibutuhkan waktu insert 25,9138 detik dan waktu *query* 24,8061 detik.
3) HBF dapat memampatkan data lebih dari 70% untuk FP>=0,001.

Referensi

[1] *N. Cheng and F. Rocca, "An Examination of the Bloom Filter and its Application in Preventing Weak Password Choices," Int. J. Comput. Appl. Technol. Res., vol. 6, no. 4, pp. 190–193, 2017, doi: 10.7753/ijcatr0604.1004.*

[2] *R. Prasetyo and A. Suryana, "Aplikasi Pengamanan Data dengan Teknik Algoritma Kriptografi AES dan Fungsi Hash SHA-1 Berbasis Desktop," J. Sisfokom (Sistem Inf. dan Komputer), vol. 5, no. 2, pp. 61–65, 2016, doi: 10.32736/sisfokom.v5i2.40.*

[3] *M. R. Irliansyah, S. D. Nasution, and K. Ulfa, "Penerapan Metode Deflate Dan Algoritma Goldbach Codes Dalam Kompresi File Teks," KOMIK (Konferensi Nas. Teknol. Inf. dan Komputer), vol. 1, no. 1, pp. 186–189, 2017.*

[4] *Supriyadi and O. Frida, "Analisis Perbandingan Pemampatan Data Teks Dengan Menggunakan Metode Huffman Dan Half – Byte," Algoritm. J. Ilmu Komput. dan Inform., vol. 2, no. 1, pp. 1–6, 2018.*

[5] *N. J. Et. al., "Data Membership Identification using Bloom Filter in Cloud Storage for Effective Resource Allocation," Inf. Technol. Ind., vol. 9, no. 2, pp. 102–108, 2021, doi: 10.17762/itii.v9i2.308.*

[6] *Y. Zhu, H. Jiang, and J. Wang, "Hierarchical Bloom Filter Arrays (HBA): A novel, scalable metadata management system for large cluster-based storage," Proc. - IEEE Int. Conf. Clust. Comput. ICCC, pp. 165–174, 2004, doi: 10.1109/CLUSTR.2004.1392614.*

[7] *S. Shomaji, F. Ganji, D. Woodard, and D. Forte, "Hierarchical Bloom Filter Framework for Security, Space-efficiency, and Rapid Query Handling in Biometric Systems," 2019 IEEE 10th Int. Conf. Biometrics Theory, Appl. Syst. BTAS 2019, no. September, 2019, doi: 10.1109/BTAS46853.2019.9185977.*

[8] *D. Gupta and S. Batra, "A short survey on bloom filter and its variants," Proceeding - IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2017, vol. 2017-Janua, pp. 1086–1092, 2017, doi: 10.1109/CCAA.2017.8229957.*

[9] *Firdaus, Z. (2019). "Implementasi Algoritma Advanced Encryption Standard (Aes) Sebagai Sistem Pengamanan Data Pengarsipan Pada Perpustakaan Digital Di Puslitbang Geologi Kelautan (Doctoral dissertation, Universitas Komputer Indonesia)".*

[10] *Erliana, C. I. (2017). "Bisnis Rental Mobil Melalui Internet (E-Commerce) Menggunakan Algoritma Sha-1 (Sequre Hash Algorithm-1)". Speed-Sentra Penelitian Engineering dan Edukasi, 4(2).*