Implementasi Hash Tree dengan Komputasi Paralel untuk Mempercepat Performa Hash dalam Ukuran Besar

Samuel Cahyawijaya / 13509082
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509082@itb.ac.id

***Abstract*— Hash merupakan salah satu teknik yang sering digunakan untuk melakukan verifikasi data. Hash dilakukan dengan mengubah nilai suatu data menjadi data lain dengan panjang tertentu yang umumnya lebih pendek dari data semula. Salah satu permasalahan dalam hash adalah *collision*, dimana terdapat nilai hash yang sama dari 2 data awal yang berbeda. Agar memiliki *collision resistance* yang tinggi, fungsi hash dibuat dengan langkah yang rumit dan dilakukan dalam beberapa iterasi, sehingga proses pembentukkan hash memerlukan waktu yang cukup lama. Dalam karya tulis yang dibuat akan diimplementasikan hash tree dengan perhitungan yang dilakukan secara paralel, tujuannya agar proses perhitungan hash dapat dilakukan paralel sehingga pembentukkan tree berlangsung lebih cepat. Dalam karya tulis ini, akan dilakukan perbandingan performa dari fungsi hash MD5 dan SHA1 dengan hash tree yang dibentuk secara paralel menggunakan fungsi hash MD5 dan SHA1. Analisis dilakukan berdasarkan analisis ruang waktu dan analisis keamanan.**

***Kata Kunci*— Hash, Hash Tree, SHA1, MD5, Paralel**

I. PENDAHULUAN

Fungsi Hash adalah fungsi satu arah yang menerima masukan *string* yang panjangnya sembarang dan mengkonversinya menjadi *string* keluaran yang memiliki panjang tetap (*fixed*) yang umumnya berukuran jauh lebih kecil daripada ukuran *string* semula [1].Sebagai contoh, fungsi hash MD5 menerima input *string* dengan panjang sembarang, dan menghasilkan *string* keluaran dengan panjang 128 bit.

Dalam pengembangannya, terdapat banyak struktur data hash yang digunakan untuk memetakan suatu data menjadi suatu nilai hash. Struktur data tersebut diantaranya, hash table (hash map), hash list, dan hash tree. Pada hash table, setiap nilai data dipetakan menjadi indeks tertentu dimana indeks tersebut akan mengacu pada sebuah nilai hash dari suatu penampung nilai hash (*buckets*). Pada hash list, hash dilakukan dalam 2 tahap, yaitu hashing pada node-node anak dan hash dari node-node anak menjadi root hash. Pada hash tree hash dilakukan sebanyak T tahap, dimana T adalah tinggi dari hash tree. Pada setiap tingkat tree, akan dihitung hash berdasarkan nilai dari node-node anak, dan pada daun nilai hash dihitung dari nilai input yang diberikan.

Dengan menggunakan struktur hash tree (disebut juga sebagai Merkle tree), pengecekan suatu data dapat dilakukan secara parsial, sehingga tidak memerlukan seluruh data terlebih dahulu untuk dapat melakukan pengecekan. Hash tree sudah digunakan untuk perubahan file pada revision control system dan untuk pengecekan validitas block yang dikirimkan pada jaringan peer to peer.

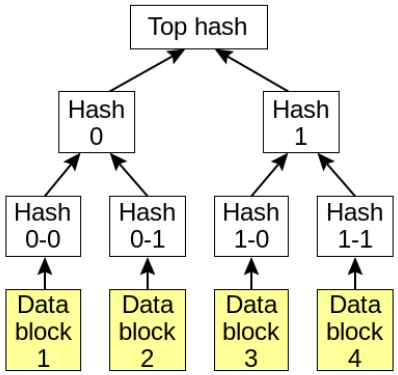
Dalam makalah ini, akan dibahas mengenai peningkatan performa dari hash tree dengan melakukan perhitungan secara paralel. Selain itu akan dilakukan analisis ruang dan waktu dan analisis keamanan fungsi hash dari implementasi hash tree yang dibangun.

.

II. DASAR TEORI

A. Hash Tree

Hash Tree adalah sebuah struktur data hash yang bertingkat dimana pada setiap node menyimpan hasil hash dari gabungan nilai pada node-node anak. Pada bagian daun hash tree, nilai hash didapat dari hasil hash partisi *data block*.



Gambar 1. Struktur Hash Tree

Dengan menggunakan hash tree, data dapat diverifikasi secara parsial, dan keseluruhan data dapat diverifikasi hanya dengan sebagian dari tree (tidak perlu memiliki seluruh node tree). Sebagai contoh, pada gambar 1, *data block* 2 dapat diverifikasi dengan hanya memiliki hash 0-1. *Data block* 2 juga dapat diverifikasi tanpa hash 0-1

dengan syarat diketahui hash 0-0, hash1, dah Top hash.

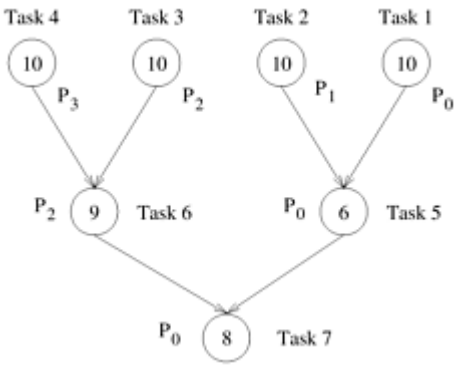
Selain itu, untuk memverifikasi seluruh data tidak diperlukan seluruh node tree. Sebagai contoh, dengan memiliki hash 0-0, hash 0, dan top hash, maka nilai dari setiap data block dapat diverifikasi, *data block 2* diverifikasi dengan membeandingan hash 0 dengan hash dari konkatenasi hash 0-0 dan hasil hash *data block 2*, *data block 3* dan *data block 4* diverifikasi dengan membandingkan top hash dengan hash dari konkatenasi dari hash 0 dengan hasil hash dari gabungan hash *data block 3* dan *data block 4*.

Jika tree yang dibangun telah dibuat minimal dengan menghilangkan beberapa node tree (dengan syarat tree yang tersisa merupakan tree minimal untuk melakukan verifikasi), maka akan terbentuk *Fractal Hash Tree.*

B. Komputasi Paralel

Komputasi paralel adalah metode perhitungan yang dilakukan secara paralel dengan memanfaatkan *multi core* pada sebuah mesin, beberapa mesin dalam sebuah jaringan, atau menggunakan GPU untuk melakukan perhitungan secara paralel (GPGPU).

Pada komputasi paralel, sebuah pekerjaan (*task*) akan didekomposisi menjadi sekumpulan pekerjaan yang akan dilakukan secara paralel. Hasil dekomposisi dari pekerjaan tersebut biasanya berbentuk struktur *tree*.

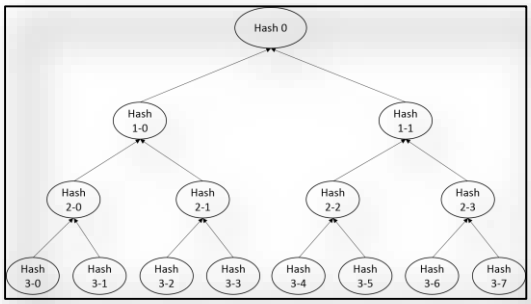


Gambar 2. Dekomposisi Pekerjaan

Setiap node pada tree merupakan sebuah pekerjaan yang telah didekomposisi. Setiap pekerjaan tersebut memiliki beban tertentu. Jika dilakukan analisis dari hasil dekomposisi pekerjaan, dapat diketahui seberapa besar derajat kekonkurenan (seberapa besar percepatan maksimal yang dapat dihasilkan) dari pekerjaan bila dilakukan secara paralel.

III. IMPLEMENTASI ALGORITMA

Implementasi hash tree yang dibuat memiliki struktur tree yang seimbang dan struktur yang statik. Hash tree dibuat seimbang agar memiliki derajat konkurensi yang tinggi. Hash tree yang dibangun terdiri dari 8 node daun dengan kedalaman 4. Fungsi hash yang



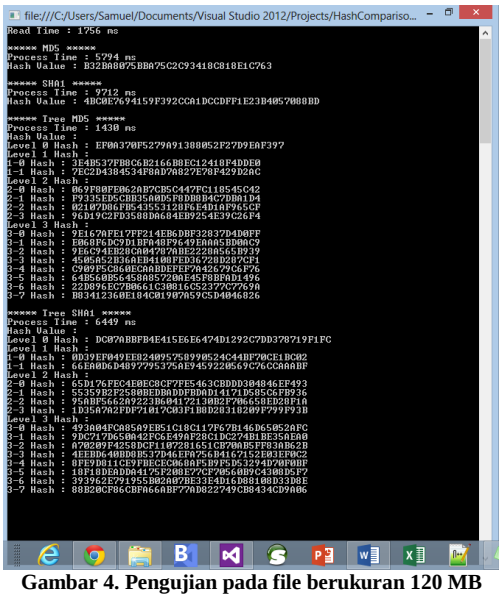
Gambar 3. Hash Tree hasil implementasi

diimplementasikan pada hash tree adalah SHA1 dan MD5.

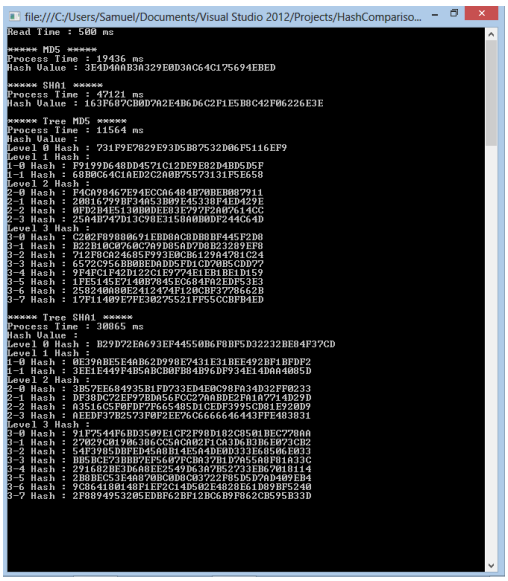
Pada daun dari hash tree, dilakukan perhitungan menggunakan SHA1 dan MD5 dari partisi data menjadi 8 bagian, pada node-node diatasnya dilakukan perhitungan hash SHA1 dan MD5 dari konkatenasi hasil hash pada node anak.

Proses paralel diimplementasi menggunakan 1 mesin dengan memanfaatkan *multi core*. Hal ini dilakukan dengan melakukan *multi threading* dalam melakukan komputasi hash pada satu level tree yang sama.

Dari hash tree yang diimplementasikan, dilakukan pengujian pada satu mesin dengan *processor* Intel i5 dengan 4 core dan clock rate 2.53 GHz dengan 8GB RAM dan sistem operasi Windows 8. Hasil dari pengujian berupa waktu eksekusi hash pada file berukuran diatas 100MB.



Gambar 4. Pengujian pada file berukuran 120 MB



Gambar 5. Pengujian pada file berukuran 450 MB

Gambar 6. Pengujian pada file berukuran 1.5 GB

IV. ANALISIS ALGORITMA

Pengujian dari algoritma yang telah diimplementasikan dilakukan dengan melakukan analisis ruang waktu dan analisis keamanan.

A. Analisis ruang waktu

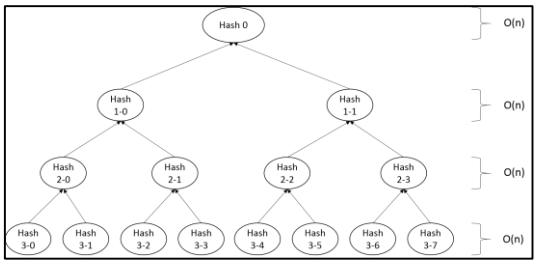
1. Ruang

Pada hash MD5 standar, besar penampung yang diperlukan adalah 128 bit. Pada hash tree yang dibangun, setiap node memiliki nilai hash MD5nya masing-masing, karena struktur hash tree yang diimplementasikan konstan, maka besar penampung yang diperlukan bernilai konstan sebesar jumlah node * 128 bit, yaitu 1920 bit

Sedangkan pada hash SHA1, besar penampung yang diperlukan adalah 160 bit. Oleh karena itu, maka besar penampung yang diperlukan sebesar jumlah node * 160 bit = 2400 bit

2. Waktu

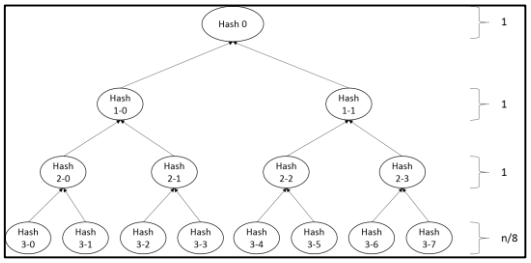
Analisis waktu dapat dilakukan dengan mengkalkulasi kompleksitas algoritma dan menghitung percepatan maksimum yang dapat dilakukan. Kompleksitas algortima dari setiap node hash tree adalah sebanding dengan nilai kompleksitas dari fungsi SHA1 dan fungsi MD5, yaitu O(n), dimana n adalah ukuran data yang di hash.



Gambar 7. Kompleksitas Algoritma setiap Node

Kompleksitas algoritma dari hash tree dihitung dengan menjumlahkan kompleksitas dari setiap node hash tree. Karena tree terdiri dari 15 node maka didapat nilai total kompleksitas 15 * n = 15n, sehingga kompleksitas algoritma dari hash tree yang diimplementasikan adalah O(n).

Besar percepatan maksimal yang dapat dicapai (derajat konkurensi) dihitung dengan pemberian beban pada setiap node seperti pada **gambar 8**.



Gambar 7. Nilai beban setiap Node

Total beban bila program dieksekusi secara sekuensial adalah $(8 * \frac{n}{8}) + (4 * 1) + (2 * 1) + 1 = n + 7$. Sedangkan eksekusi terpanjang (*critical path*) dari eksekusi secara paralel adalah $\frac{n}{8} + 1 + 1 + 1 = \frac{n}{8} + 3$. Derajat konkurensi = total beban / critical path = $(n + 7) / (\frac{n}{8} + 3)$, dengan n adalah besar data yang diproses. Bila n cukup besar, maka derajat konkurensi akan mendekati 8.

B. Analisis Keamanan

Jumlah bit yang dihasilkan mencapai 1920 bit (untuk MD5) dan 2400 bit (untuk SHA1), dalam hash tree yang dibangun setiap node terdiri dari 1 nilai hash dengan fungsi hash tertentu, sehingga

kemungkinan terjadinya *collision* dalam 1 node sama dengan kemungkinan terjadinya *collision* pada algoritma hash yang digunakan.

Namun untuk pengecekan node selain daun dapat dilakukan dengan 2 solusi, yaitu dengan melakukan pengecekan nilai hash pada node tersebut (lebih efisien) atau dengan melakukan pengecekan nilai hash pada node tersebut dan seluruh nilai hash dari anak node tersebut (lebih aman). Bila pengecekan dilakukan berdasarkan nilai hash pada node tersebut, maka kemungkinan terjadinya *collision* sebanding dengan *collision* pada algoritma hashnya, sedangkan bila dihitung berdasarkan anak-anaknya, maka probabilitas terjadinya *collision* akan menjadi probabilitas *collision* fungsi hash dipangkatkan seluruh node yang dihitung hashnya (node tersebut dan node-node anak dari node tersebut).

V. KESIMPULAN

1. Hash tree yang diimplementasikan memerlukan ukuran penyimpanan yang lebih besar, yaitu 1920 bit untuk fungsi hash MD5 dan 2400 bit untuk fungsi hash SHA1.
2. Hash tree yang diimplementasikan memiliki kompleksitas algoritma O(n) dan derajat konkurensi (n + 7) / ($\frac{n}{8}$ + 3).
3. Probabilitas *collision* pada setiap node daun hash tree sama dengan probabilitas *collision* dari fungsi hashnya.
4. Probabilitas *collision* pada setiap node selain daun pada hash tree dapat bernilai minimal dengan probabilitas *collision* dari fungsi hashnya dan maksimal bernilai probabilitas *collision* dari fungsi hash dipangkatkan dengan jumlah node seluruh node yang dihitung hashnya (node tersebut dan node-node anak dari node tersebut).

REFERENCES

[1] Munir, Rinaldi. "Diktat Kuliah IF3054 Kriptografi". *Departemen Teknik Informatika Institut Teknologi Bandung.*, 2005.

[2] http://www.rohitab.com/discuss/topic/31377-md5-vs-sha-1-how-to-calculate-the-algorithm-complexity, tanggal akses 19 Mei 2013

[3] Ederov, Boris. "Merkle Tree Traversal Techniques". *Department of Computer Science Darmstadt University oh Technology*. 2007

[4] Markus Jackobson, Tom Leighton, Silvio Micali, dan Michael Szydlo. "Fractal Merkle Tree Representation and Traversal". *RSA Laboratories, MIT Laboratoy of Computer Science, Akamal Technologies.*2003

[5] Wesley, Addison. "Introduction to Parallel Computing". *The Benjamin/Cummings Publishing Company, Inc* .2003

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Mei 2013

Samuel Cahyawijaya
13509082