Penggunaan Nilai Hash untuk Sinkronisasi Direktori pada Layanan *Cloud Storage*

Rita Wijaya (13509098)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509098@std.stei.itb.ac.id

Abstract*—Cloud storage kini menjadi trend baru dalam dunia teknologi informasi. Penyedia layanan maupun pengguna dari cloud storage semakin bertambah jumlahnya. Salah satu fitur yang banyak dimanfaatkan dari cloud storage ini adalah fitur sinkronisasi direktori. Pengguna layanan dapat mengakses sebuah direktori di mana saja dan kapan saja. Apabila direktori mengalami perubahan, maka direktori yang sama juga akan berubah meskipun diakses dari perangkat yang lain. Salah satu kendala dalam sinkronisasi ini adalah proses pengecekannya. Teknik pengecekan direktori yang dapat digunakan adalah teknik pengecekan menggunakan nilai hash. Pada makalah ini, akan dibahas teknik pengecekan direktori menggunakan nilai hash dan dibandingkan dengan teknik pengecekan konvensional.

***Index Terms*— cloud storage, hash, sinkronisasi**

I. PENDAHULUAN

Kini, penyedia *cloud storage* atau penyimpanan berkas secara *online* semakin bertambah jumlahnya. Pengguna layanan tersebut dapat menyimpan berkas yang dimilikinya di server sehingga berkas tersebut dapat diakses kapan saja dan di mana saja. Salah satu fitur yang ditawarkan layanan *cloud storage* adalah sinkronisasi direktori. Layanan ini memungkinkan pengguna memiliki direktori pada server yang sama persis dengan suatu direktori pada komputer pribadi.

Dalam melakukan sinkronisasi direktori, terdapat beberapa proses yang harus dilakukan. Salah satu proses yang ada adalah proses perbandingan isi kedua direktori. Pada proses ini, akan dilakukan identifikasi direktori dan berkas apa saja yang berbeda. Apabila terdapat perbedaan, maka berkas di server harus segera diperbaharui sesuai dengan kondisi berkas saat ini.

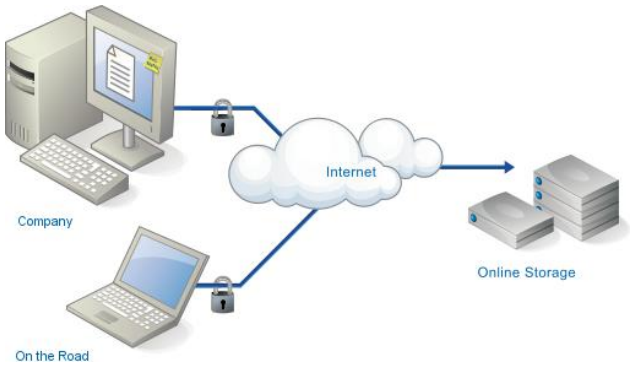
Salah satu cara identifikasi perbedaan pada direktori adalah dengan pengecekan isi direktori secara terus-menerus. Apabiladitemukan perbedaan pada direktori, maka berkas lama akan diperbaharui sehingga konsistensi direktori terjaga. Salah satu permasalahan dalam sinkronisasi direktori adalah dalam direktori mungkin saja terdapat direktori lainnya. Berkas yang ada dalam suatu direktori juga tidak tentu jumlahnya dengan ukuran yang bervariasi. Untuk memastikan bahwa kedua berkas memiliki isi yang sama saja kita perlu memastikan bahwa seluruh string bit penyusun berkas sama. Apabila sinkronisasi dilakukan secara lokal, maka pengecekan ini dapat dilakukan dengan lebih mudah dan cepat. Namun, untuk sinkronisasi dengan server, diperlukan waktu dan bandwidth lebih untuk mengirimkan isi suatu berkas. Sinkronisasi direktori menjadi infeasible apabila pertukaran berkas harus dilakukan terus-menerus.

Nilai hash merupakan salah satu alternatif untuk mengecek adanya perbedaan pada berkas. Pada makalah ini akan dibahas tentang cara identifikasi direktori dan berkas menggunakan nilai hash. Kinerja perbandingan nilai hash ini akan dibandingkan dengan kinerja perbandingan konvensional.

II. LANDASAN TEORI

A. Cloud Storage

Penyimpanan data pada server *cloud* yang dapat diakses secara *online* disebut sebagai *cloud storage*. Berkas yang dimiliki dapat disimpan pada sehingga pengguna dapat mengakses berkas yang sama secara *online*. Banyak penyedia jasa *cloud storage* yang ada, di antaranya Justcloud.com, Zip Cloud, Dropbox, dan myPCBackup.com.



Gambar 1. Skema *cloud storage*
(sumber: http://www.williamburdine.com/wp-content/uploads/2013/04/the-cloud.png)

Dibandingkan dengan metode penyimpanan tradisional, *cloud storage* memiliki kelebihan sebagai berikut:

- Perusahaan tidak perlu menginsatalasi *physical storage* pada pusat data atau kantor lagi.
- Pemeliharaan *storage* seperti *backup* atau penambahan *physical storage* akan menjadi tanggung jawab penyedia layanan *cloud storage*.
- Perusahaan hanya perlu membayar untuk *storage* yang benar-benar digunakan saja.

Beberapa karakteristik yang dimiliki *cloud storage* adalah *performance*, *manageability*, dan *availability* [1].

B.Fungsi Hash

Fungsi hash merupakan algoritma satu arah yang memetakan string data yang panjangnya beragam menjadi nilai yang memiliki panjangnya tetap [2]. Nilai yang dikembalikan oleh fungsi hash disebut sebagai nilai hash. Input yang sama harus menghasilkan nilai hash yang sama. Nilai hash yang sama harus dihasilkan dari input yang sama pula. Apabila inputnya berbeda, nilai hash yang dihasilkan haruslah berbeda. Apabila input yang berbeda memiliki nilai hash yang sama, maka situasi tersebut disebut *collision*. Fungsi hash yang bebas *collision* adalah fungshi hash h yang memenuhi syarat berikut [3]:

1. Argumen X dapat memiliki panjang sembarang dan hasil dari h(X) memiliki panjang tetap.
2. Fungsi hash harus bersifat searah. Apabila diketahui nilai Y yang merupakan hasil dari h, maka akan sulit (*infeasible*) untuk menemukan X yang memenuhi h(X) = Y. Apabila diketahui X dan h(X), maka akan sulit (*infeasible*) untuk menemukan X' yang X' ≠ X dan h(X') = h(X).
3. Sulit (*infeasible*) untuk menemukan 2 pesan berbeda yang memiliki nilai hash yang sama.

Nilai hash sangat sensitif terhadap nilai masukannya. Perbedaan kecil pada masukan akan menghasilkan nilai hash yang jauh berbeda. Oleh sebab itu, fungsi hash umum digunakan untuk verifikasi integritas pesan, verifikasi *password*, atau sebagai *identifier* dari data. Algoritma hash yang ada antara lain MD2, MD4, MD5, SHA-0, SHA-1, dan RIPEMD.

C. SHA-1

Fungsi hash SHA-1 merupakan anggota keluarga SHA (*Secure Hash Algorithm*). Fungsi hash SHA-1 didesain oleh United States National Security Agency dan dipublikasikan oleh NIST pada tahun 1995 [4]. SHA-1 termasuk fungsi hash yang umum digunakan kini. SHA-1 menerima pesan dengan panjang kurang dari 2^{64} bit dan menghasilkan nilai hash sepanjang 160 bit.

Pertama-tama, pesan masukan di-*padding* dan diproses dalam blok berukuran 512 bit dengan struktur iteratif Damgard/Merkle. Masing-masing iterasi memanggil sebuah fungsi kompresi yang menerima masukan sebuah *chaining value* sepanjang 160 bit dan sebuah blok pesan berukuran 512 bit. Fungsi kompresi tersebut akan mengembalikan sebuah *chaining value* sepanjang 160 bit. *Chaining value* awal yang disebut sebagai *initial value* (IV) merupakan sebuah konstanta yang telah ditetapkan. Nilai *chaining value* terakhir yang menjadi nilai hash dari pesan masukan

Berikut akan dijelaskan fungsi kompresi dari SHA-1 [5]. Bagi masing-masing blok pesan berukuran 512 bit menjadi 16 blok berukuran 32 bit (m_0, m_1, m_2, ..., m_{15}). Berikutnya, untuk i = 16 hingga 79,

$$m_i = m_{1-3} \oplus m_{1-8} \oplus m_{1-14} \oplus m_{1-16}) \lll 1 .$$

Pesan yang diperoleh kemudian diproses dalam 4 putaran, masing-masing putaran tersebut terdiri dari 20 langkah. Untuk 1 = 1, 2, ..., 80,

$a_i = (a_{i-1} \lll 5) + f_i(b_{i-1}, c_{i-1}, d_{i-1}) + e_{i-1} + m_{i-1} + k$
$b_i = a_{i-1}$
$c_i = b_{i-1} \lll 30$
$d_i = c_{i-1}$
$e_i = d_{i-1}$

Initial chaining value ditetapkan sebagai:

$IV = (a_0, b_0, c_0, d_0, e_0)$;
$IV = (0x67452301, 0xefcdab89, 0x98badcfe,$
$0x10325476, 0xc3d2e1f0)$

Setiap putaran yang ada menggunakan fungsi Booolean f_i dan konstanta k_i yang terdefinisi pada Tabel 1.

Tabel 1. Fungsi Boolean dan konstanta SHA-1

round	iterasi	f_i	k_i
1	1-20	IF: $(x \wedge y) \vee (-x \wedge z)$	0x5a827999
2	21-40	XOR: $x \oplus y \oplus z$	0x6ed6eba1
3	41-60	MAJ: $(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$	0x8fabbcdc
4	61-80	XOR: $x \oplus y \oplus z$	0xca62c1d6

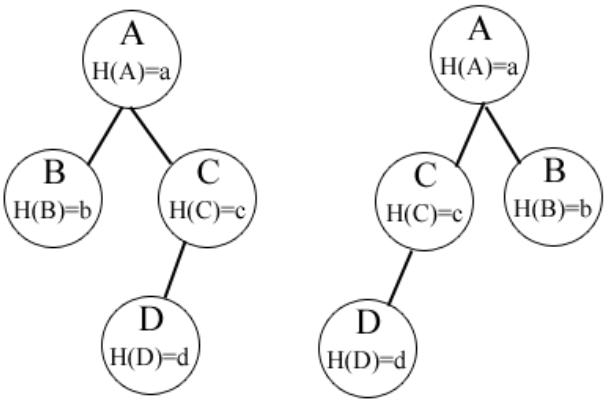
III. PENGECEKAN DIREKTORI DENGAN FUNGSI HASH

Pada bagian ini akan dijelaskan tentang teknik pengecekan direktori menggunakan fungsi hash dengan struktur pohon. Pembentukan pohon terdiri dari2 tahap, yaitu penyusunan node dan penghitungan nilai hash. Proses identifikasi direktori dilakukan dengan membandingkan struktur kedua pohon. Berikut akan dijelaskan lebih rinci dari masing-masing tahapan dalam identifikasi direktori ini.

A.Pembentukan Pohon

Struktur pohon akan menjadi struktur data yang digunakan untuk menyimpan nilai hash dari setiap direktori dan berkas pada direktori yang akan disinkronisasi. Pohon ini merupakan pohon n-ary yang setiap node penyusunnya merupakan direktori atau

berkas. Direktori utama yang akan disinkronisasi akan menjadi akar. Node lain disusun dari direktori di bawahnya. Sedangkan, direktori kosong dan berkas akan menjadi dari pohon.



Gambar 2. Pohon identik

Untuk membentuk pohon, langkah-langkah yang dilakukan adalah sebagai berikut:

1. Menginisialisasi sebuah pohon kosong.
2. Mengisi akar pohon dengan direktori utama.
3. Untuk setiap isi f di dalam direktori, lakukan:
    a. Menyimpan f sebagai sebuah node.
    b. Apabila f merupakan direktori, ulangi langkah 3 dengan f sebagai direktori acuan.

Selanjutnya, pohon yang telah terbentuk ini akan digunakan untuk menyimpan informasi nilai hash direktori.

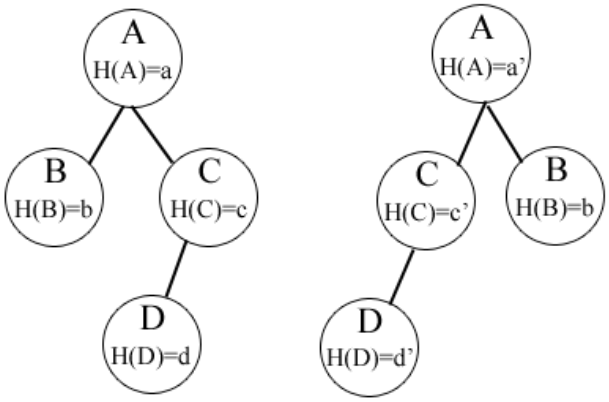
B.Penghitungan Nilai Hash

Setiap node pohon yang terbentuk akan dihitung nilai hash-nya. Penghitungan nilai hash dilakukan dari daun hingga akar. Penghitungan nilai hash ini dilakukan dengan algoritma rekursif berikut:

1. jadikan akar sebagai *current node*
2. Untuk setiap anak a dari *current node*, lakukan:
    a. Apabila a merupakan sebuah berkas, hitung nilai hash dari isi berkas tersebut
    b. Apabila b merupakan sebuah direktori, hitung nilai hash direktori dengan masukan berupa nilai seluruh hash anak-anaknya.

C.Pembandingan Pohon

Kedua direktori dianggap telah tersinkronisasi apabila kedua direktori memiliki struktur pohon yang sama dengan nilai hash pada masing-masing node yang sama pula. Ilustrasi pohon dengan node dan nilai hash identik dapat dilihat pada Gambar 2. Apabila nilai hash pada suatu node sama, maka itu berarti node dan seluruh isi di bawah node tersebut juga sama. Perubahan suatu berkas dapat atau direktori dapat mempengaruhi nilai hash node di atasnya seperti yang diilustrasikan pada Gambar 3. Apabila nilai hash pada suatu node berbeda, berarti kesamaan isi node dan seluruh node di bawahnya dipertanyakann.



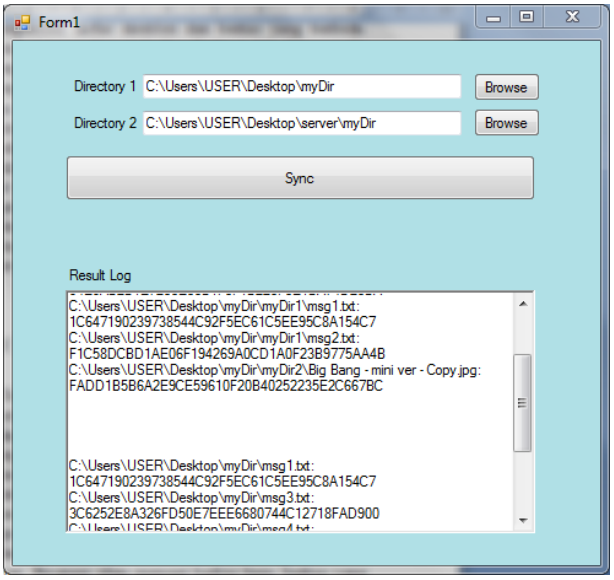
Gambar 3. Pohon setelah berkas mengalami perubahan

Misalkan A merupakan pohon yang dibentuk dari direktori komputer dan B merupakan pohon yang dibentuk dari direktori server. Maka untuk membandingkan kedua pohon tersebut, lakukan langkah berikut:

1. Jadikan akar A dan B sebagai *current node-a* dan *current node-a*.
2. Apabila nilah hash *current node-a* dan *current node-b* sama, maka proses pembandingan diterminasi.
3. Untuk setiap anak *current node-a* yang bukan anak *current node-b*, artinya anak dari *current node-a* tersebut belum ada di server
4. Apabila *current node-b* bukan daun, maka untuk seluruh anak b pada *current node-b* lakukan:
    a. Apabila b bukan anak dari *current node-a*, artinya b telah dihapus dari direktori komputer
    b. Apabila b merupakan anak dari *current node-a* juga, jadikan a dan b sebagai *current node-a* dan *current node-b* dan ulangi langkah 2
5. Apabila *current node-b* merupakan daun, maka pindahkan isi dari *current node-a* mengalami perubahan

Dari proses pembandingan pohon, kita berhasil memperoleh daftar direktori dan berkas yang berbeda. Untuk menyamakan isi kedua direktori, kita hanya perlu mengirimkan nama berkas dan direktori yang perlu dihapus saja. Selanjutnya, hanya direktori dan berkas baru atau yang mengalami perubahan saja yang perlu dikirimkan seluruhnya. Dari sini, dapat kita lihat bahwa jumlah pertukaran berkas menjadi berkurang. Pada bagian berikutnya akan dibahas lebih lanjut tentang implementasi dari teknik ini. Kemudian, akan dikaji apakah teknik ini

dapat berjalan dengan baik untuk identifikasi isi direktori. Teknik ini juga akan dianalisis kelebihan dan kekurangannya dibandingkan teknik identifikasi konvensional.



Gambar 4. Tampilan program beserta hasil eksekusinya

IV. IMPLEMENTASI, PENGUJIAN, DAN ANALISIS

Teknik sinkronisasi direktori yang telah dipaparkan sebelumnya akhirnya diimplementasikan menggunakan bahasa pemrograman C#. Perangkat lunak ini dibangun menggunakan sistem operasi Windows 7 Professional. Perangkat lunak ini bertujuan untuk mensimulasikan teknik identifikasi direktori. Program akan menerima masukan berupa 2 direktori. Salah satu direktori akan berperan sebagai direktori pada komputer pengguna, sedangkan direktori kedua berperan sebagai direktori server. Program akan mencari berkas baru, berkas yang dihapus, dan berkas yang mengalami perubahan dari direktori komputer pengguna. Contoh tampilan program dan hasil eksekusinya dapat dilihat pada Gambar 4.

Pada seluruh pengujian yang dilakukan, teknik pengecekan direktori menggunakan nilai hash ini berhasil mengidentifikasi seluruh direktori dengan tepat. Berikut potongan dari log hash yang diperoleh dan hasil identifikasinya:

```
Root: \myDir\
\myDir\msg1.txt:
1C647190239738544C92F5EC61C5EE95C8A154C7
...
...
\myDir\msg7.txt:
BDA90DC0D0383AFC3B60A264DBE110F34A403217
\myDir\msg8.txt:
54E5AB2B1E7280E08B1F0F1B228F8E1BA74D20BA
...
```

```
Root: \server\myDir\
\server\myDir\msg1.txt:
1C647190239738544C92F5EC61C5EE95C8A154C7
...
...
\server\myDir\msg7.txt:
3C6252E8A326FD50E7EEE6680744C12718FAD900
\server\myDir\msg8.txt:
54E5AB2B1E7280E08B1F0F1B228F8E1BA74D20BA
...

Hasil:
...
\myDir\msg7.txt berubah
...
```

Saat pengujian dilakukan, sebagian isi dari direktori utama dimodifikasi, termasuk msg.txt. Berdasarkan hasil potongan log di atas, dapat dilihat bahwa program berhasil mengidentifikasi bahwa msg7.txt mengalami modifikasi karena nilai hash yang diperoleh berbeda.

Pengujian berikutnya yang dilakukan adalah pengujian performansi. Performansi diukur berdasarkan waktu yang dibutuhkan selama eksekusi fungsi. Seluruh pengujian dilakukan menggunakan lingkungan eksekusi yang sama.

Lama pembangunan sebuah pohon dan pengisian nilai hash bervariasi bergantung ukuran direktori. Fungsi Hash yang digunakan selama implementasi adalah fungsi SHA-1. Dari pengujian yang dilakukan, lama pembangunan pohon dan pengisian nilai hash dapat dilihat pada Tabel 2. Setiap kasus pengujian diulang sebanyak 5 kali dan diambil waktu rata-ratanya. Hal ini dilakukan untuk mengimbangi *overhead* saat pembacaan berkas untuk pertama kalinya.

Tabel 2. Lama pembangunan pohon

No	Ukuran direktori	Waktu rata-rata (ms)
1	7.28 MB	1464.8529
2	10 MB	1901.91768
3	15 MB	2359.53548
4	23.0 MB	5209.91062
5	42.5 MB	8060.14122
6	73.4 MB	14969.08266
7	85.0 MB	17400.9856

Pengujian berikutnya mengukur lamanya perbandingan kedua pohon. Lama perbandingan pohon dipengaruhi jumlah direktori dan berkas yang berbeda. Dari pengujian yang dilakukan, diperoleh rata-rata waktu perbandingan yang diperlukan cukup cepat.

Sebagai pembanding, diimplementasikan juga sistem

identifikasi direktori yang membandingkan direktori dan berkas secara konvensional. Fitur perangkat lunak yang dibangun sama dengan implementasi sebelumnya, hanya saja teknik pengecekan berkas dan direktorinya berbeda. Setiap direktori akan ditelusuri isinya. Pada teknik pengecekan konvensional ini, seluruh isi direktori akan ditelusuri. Berkas yang ada akan dikirimkan untuk dibandingkan isinya. Pengecekan isi berkas sendiri dilakukan per blok. Setiap blok harus tersusun dari string bit yang sama persis. Lama pengecekan tiap berkas bergantung pada ukuran berkas dan letak perbedaan yang ada. Apabila kedua berkas sama persis, sistem tetap harus meneruskan pengecekan hingga akhir berkas. Performansi dari teknik pengecekan ini diukur dengan cara yang sama dengan pengukuran sebelumnya. Masing-masing kasus pengujian diulangi sebanyak 5 kali dan diambil waktu rata-ratanya.

Tabel 3. Lama perbandingan 2 pohon

No	Jumlah direktori		Jumlah berkas		Jumlah node		Waktu (ms)
	pengguna	server	pengguna	server	pengguna	server	
1	5	5	7	7	12	12	4.042
2	6	5	9	6	15	11	4.729
3	3	4	22	20	25	24	4.671
4	7	7	45	45	52	52	4.311
5	10	10	72	72	82	82	4.562
6	12	12	53	52	65	64	4.323

Apabila hanya memperhatikan waktu pengecekannya saja, tanpa memperhitungkan waktu pembentukan pohon hash, maka waktu yang diperlukan oleh teknik pengecekan hash jauh lebih cepat dibandingkan pengecekan konvensional. Hal ini disebabkan karena untuk membandingkan kedua berkas, sistem cukup membandingkan nilai hash yang panjang tetap, yaitu 160 bit saja. Sedangkan teknik pengecekan konvensional memerlukan waktu lebih lama karena sistem perlu membaca isi kedua berkas dan membandingkan isi keduanya. Apabila berkas berukuran lebih besar, maka kemungkinan waktu pengecekannya juga akan lebih lama.

Meskipun waktu pengecekannya lebih cepat, teknik pengecekan menggunakan nilai hash ini juga memiliki kelemahan. Waktu yang diperlukan untuk menghitung nilah hash saat pembentukan pohon cukup berbanding lurus dengan ukuran direktori yang ada. Apabila direktori berukuran besar, maka semakin lama waktu yang dibutuhkan untuk membangun sebuah pohon.

Secara keseluruhan, dengan memperhitungkan waktu yang dibutuhkan untuk menghitung nilai hash, maka teknik pengecekan direktori dengan menggunakan nilai hash ini memerlukan waktu yang lebih lama dibandingkan teknik pengecekan konvensional. Meskipun lebih lama, bukan berarti performansi yang dimiliki selalu lebih buruk. Teknik pengecekan menggunakan nilai hash efektif digunakan apabila direktori jarang mengalami perubahan. Dengan demikian, pohon tidak harus selalu dibuat dari awal. Pada kasus ini, waktu pengecekan yang diperlukan menjadi jauh lebih cepat, yaitu sama dengan waktu pembandingan 2 pohon saja.

Tabel 4. Lama pengecekan konvensional

No	Ukuran direktori	Waktu rata-rata (ms)
1	6.52 MB	205.0201
2	54.6 MB	1834.6309
3	71.0 MB	2157.0759
4	73.4 MB	2626.6633
5	90 MB	3167.3076
6	150 MB	5230.2203

Ketika diaplikasikan pada layanan sinkronisasi direktori *cloud storage*, teknik pengecekan menggunakan nilai hash memiliki kelebihan. Lokasi komputer pengguna dan lokasi server yang berjauhan membuat pertukaran informasi keduanya memerlukan waktu yang lebih lama. Pada teknik pengecekan konvensional, waktu pengiriman informasi direktori berbanding lurus dengan ukuran direktori. Dengan teknik pengecekan menggunakan nilai hash, sistem tidak perlu mengirimkan seluruh isi berkas, cukup pohon dengan nilai hash saja yang perlu dikirimkan. Bayangkan apabila direktori berisi banyak berkas besar, maka perbedaan ukurannya akan sangat besar. Setelah melalui proses identifikasi, hanya berkas yang benar-benar baru dan berbeda saja yang perlu dikirimkan. Dengan demikian, *bandwidth* dan waktu yang dibutuhkan dapat dikurangi.

V. SIMPULAN

Fungsi hash dapat diaplikasikan untuk identifikasi berkas dan direktori dalam proses sinkronisasi direktori. Kelebihan dari teknik ini adalah waktu pengecekan yang dibutuhkan lebih sedikit dibandingkan teknik pengecekan konvensional yang memeriksa isi berkas satu per satu. Setelah pengecekan berhasil dilakukan, cukup berkas yang diperlukan saja yang perlu dikirimkan. Hal ini dapat menghemat waktu sekaligus *bandwidth*. Namun, teknik ini juga memiliki kelemahan yaitu waktu yang dibutuhkan untuk membentuk pohon berbanding lurus dengan ukuran direktorinya. Apabila ukuran direktori besar, maka waktu yang dibutuhkan untuk menghitung nilai hash yang ada juga lebih lama.

Teknik pengecekan direktori dengan nilai hash unggul diaplikasikan pada sistem yang lokasi direktorinya terpisah dengan kondisi direktori yang jarang terjadi perubahan. Apabila sinkronisasi direktori dilakukan pada lokasi penyimpanan lokal dan isinya sering mengalami perubahan, maka disarankan untuk menggunakan teknik pengecekan konvensional saja.

DAFTAR REFERENSI

[1] S. Eswaran and D. S. Abburu, "Identifying Data Integrity in the Cloud Storage," *International Journal of Computer Science Issues*, vol. 9, no. 2, March 2012.

[2] I. B. Damgård, "A Design Principle for Hash Functions," in *CRYPTO '89 Proceedings on Advances in Cryptology*, New York, 1989.

[3] I. B. Damgard, "Collision Free Hash Functions and Public Key Signature Schemes," in *EUROCRYPT'87 Proceedings of the 6th Annuall Internarional Conference on Theory and Application of Cryptographic Techniques*, Springer-VErlag Berlin, Heidelberg, 1988.

[4] NIST, *Secure Hash Standard*, Federal Information Processing Standard, 1995.

[5] Y. L. Y. H. Y. Xiaoyun Wang, "Finding Collisions in the Full SHA-1," in *Proceedings of Crypto*, 2005.

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, 19 Mei 2013

Rita Wijaya (13509098)