SINKRONISASI FILE PADA MULTI-SERVER MENGGUNAKAN AGLETS

Faiq Wildana[*)], Maman Somantri, and Adian Fatchur Rochim

Jurusan Teknik Elektro, Universitas Diponegoro Semarang
Jl. Prof. Sudharto, SH, Kampus UNDIP Tembalang, Semarang 50275, Indonesia

*)Email : faiq.wildana@gmail.com

Abstrak

Berbagai penyebab memungkinkan terjadinya masalah pada server tunggal, sehingga perlu dipertimbangkan untuk menggunakan backup server. Backup server berisi data yang sama dengan data dari server utama. Hal tersebut dapat dilakukan dengan cara mensinkronkan file secara berkala. Dalam studi ini, sinkronisasi file diwujudkan menggunakan Aglets. Aglets merupakan framework mobile agent dengan kemampuan berpindah dan melakukan tugas sesuai keinginan. Hasil pengujian menunjukkan sinkronisasi berjalan dengan baik menggunakan prinsip mobile agent.

Kata kunci : Backup server, Mobile Agent, Aglets, Sinkronisasi file.

Abstract

Various possible causes problems in single server. It should be considered to use backup server. Backup server contains the same data with the data from the main server. This can be done by periodically synchronize files. In this study, file synchronization is realized using Aglets. Aglets is a mobile agent framework with the ability to move and perform tasks as desired. Test results indicate synchronization runs fine using the principles of mobile agents.

Keywords : Backup server, Mobile Agent, Aglets, File synchronization.

1. Pendahuluan

Ketika kita bekerja untuk membangun suatu layanan publik, kita akan selalu berhubungan dengan minimal satu server. Dengan berbagai penyebab yang memungkinkan terjadinya masalah pada layanan publik maka perlu dipertimbangkan untuk menggunakan *backup* server. *Backup* server berisi data yang sama dengan data dari server utama dan selalu *up-to-date*. Sehingga apabila terjadi masalah pada server utama masih memiliki data cadangan terbaru yang siap digunakan sesuai kebutuhan. Mekanisme penyamaan data (*data mirroring*) ini membutuhkan aplikasi sendiri, yaitu sinkronisasi *file*. Sinkronisasi *file* ini perlu dijalankan secara berkala agar memiliki data cadangan yang terbaru pada *backup* server.

Aglets merupakan Teknologi *Mobile Agent* yang memiliki kemampuan berpindah dari satu *host* ke *host* lain. Ketika berpindah, *state* program ini disimpan, dibawa ke *host* selanjutnya dan berjalan sebagai proses yang normal. Dengan kemampuan tersebut *agent* dapat digunakan untuk melakukan pekerjaan secara otomatis sesuai dengan kebutuhan. Dalam hal ini digunakan sebagai media sinkronisasi *file* antara dua server.

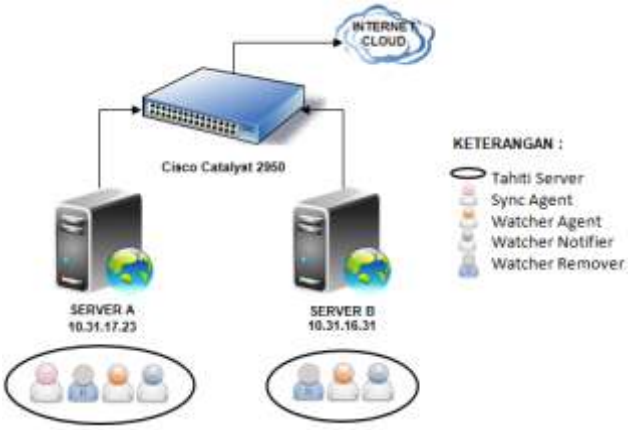
Beberapa riset yang telah dilakukan sebelumnya mengenai *Mobile Agent* seperti *The Use Of Mobile Agents In Network Management Applications* oleh Marcus Naylor [1], kemudian MAgNET : *Mobile Agents for Networked Electronic Trading* oleh Prithviraj Dasgupta dkk. [2] adalah contoh penerapan *Mobile Agent* Aglets pada *e-commerce* dimana *agent* dikirim ke berbagai pemasok, kembali dengan penawaran terbaik dan selanjutnya meminta persetujuan dari pengguna untuk melakukan transaksi tersebut atau tidak. Contoh lainnya adalah penggunaan Aglets dan JACOB untuk polling halaman web yang dibuat oleh Intan dan Joko [3] dimana *agent* merekam URL yang berguna (*useful*) bagi pengguna dan membagikannya kepada pengguna lain. Ada juga yang menggunakan *agent* untuk mendapatkan informasi dimana buku dijual pada dua atau lebih toko buku yang dilakukan oleh Scifo Anggi. [4]

2. Metode

Aplikasi sinkronisasi *file* ini merupakan aplikasi yang bekerja dengan cara menyamakan isi dari direktori yang telah ditentukan pada Server A dengan isi direktori yang telah ditentukan pada Server B. Server A bertindak sebagai server utama sedangkan Server B sebagai *backup* server. *Graphical User Interface* (GUI) dari aplikasi ini

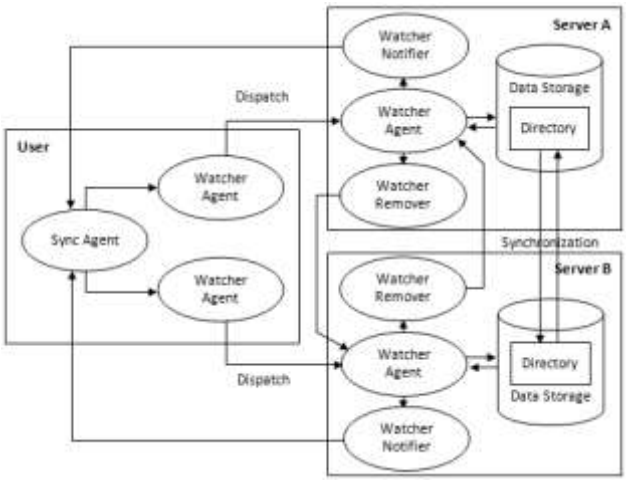
memiliki satu tampilan dari *SyncAgent*. Sedangkan untuk *agent* yang lain tidak ada. Kemudian untuk notifikasi ditampilkan di *command line*. Selain GUI dari *SyncAgent* ada juga Auto Index PHP Script yang digunakan memudahkan *user* untuk melihat daftar *file* yang disinkronkan. Auto Index PHP Script merupakan sistem manajemen konten (CMS) *open source* berbasis web yang dikembangkan menggunakan PHP oleh Justin Hagstorm. CMS ini digunakan untuk mengindeks *file* dan juga dapat mengunduhnya sesuai keinginan.

2.1. Pemodelan Topologi dan Arsitektur



Gambar 1. Topologi aplikasi sinkronisasi *file*

Topologi pada Gambar 1 merupakan dasar pengujian aplikasi dimana antara Server A dan B berada pada jaringan yang berbeda dihubungkan oleh Cisco Catalyst 2950. Mengacu pada gambar tersebut kerja agen dapat dijelaskan sebagai berikut. Server A merupakan tempat *SyncAgent* dibuat. *WatcherAgent* pada Server A dan Server B dibuat dan dikirimkan oleh *SyncAgent*. Setelah *WatcherAgent* sudah dikirimkan ke kedua server, kedua *WatcherAgent* tersebut akan berkomunikasi untuk mensinkronkan data. Apabila *WatcherAgent* mendeteksi ada *file* yang terhapus, maka *WatcherAgent* akan mengirimkan *WatcherRemover* kepada *WatcherAgent* lainnya. Informasi selama proses sinkronisasi dikirimkan oleh *WatcherNotifier* kepada *SyncAgent*. Media untuk menjalankan *SyncAgent*, *WatcherAgent* dan *WatcherNotifier* adalah Tahiti Server yang dimiliki oleh Aglets.



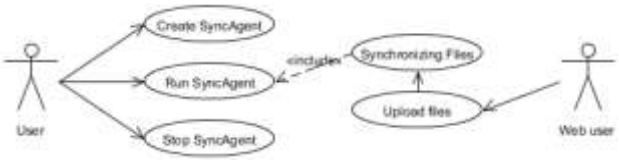
Gambar 2. Arsitektur aplikasi sinkronisasi *file*

Arsitektur dari aplikasi ini dapat dilihat pada Gambar 2. Di Gambar 2 diperlihatkan bagaimana *agent* dibuat, dikirim kemudian bekerja sama untuk mensinkronkan *file*. Pada *User*, *SyncAgent* membuat dua *WatcherAgent* kemudian masing-masing dikirimkan ke Server A dan Server B. Pada Server A dan Server B masing-masing *WatcherAgent* mengakses data yang berada di dalam direktori target kemudian saling bekerja sama untuk mensinkronkan data. Apabila *WatcherAgent* mendeteksi ada *file* yang terhapus, maka *WatcherAgent* akan mengirimkan *WatcherRemover* kepada *WatcherAgent* lainnya. Informasi selama proses sinkronisasi akan dikirimkan oleh *WatcherNotifier*.

2.2. Pemodelan UML

Activity Diagram pada Gambar 3 menunjukkan dimana proses awal sampai akhir yang harus dilakukan untuk menjalankan aplikasi sinkronisasi *file* melalui Tahiti Server. Berdasarkan Gambar 3 dapat dijelaskan sebagai berikut. Aplikasi dimulai dengan cara menjalankan Aglets. Kemudian membuat *SyncAgent*. Setelah SyncAgent dibuat maka *user* perlu mengisi form dengan benar. Apabila ada kesalahan *user* harus membenarkannya agar *SyncAgent* dapat berjalan. Apabila sudah benar *WatcherAgent* akan dikirim untuk mengecek apakah direktori untuk sinkronisasi ada. Apabila direktori tidak ditemukan maka sinkronisasi tidak dapat berlangsung. Apabila direktori ada maka proses sinkronisasi akan berlangsung. Apabila sinkronisasi gagal maka sinkronisasi akan berhenti. Apabila kerja SyncAgent dihentikan maka sinkronisasi juga akan berhenti.

Gambar 3. ***Activity Diagram*** **aplikasi sinkronisasi** ***file***



Gambar 4. ***Use Case Diagram*** **aplikasi sinkronisasi** ***file***

Tabel 1. Identifikasi ***Use Case***

No	Aktor	Use Case	Deskripsi
1.	*User*	*Create SyncAgent*	Pengguna Membuat *(create agent) SyncAgent melalui Window* tahiti : The Aglets Viewer yang dimiliki Aglets.
2.	*User*	*Run SyncAgent*	Pengguna menjalankan *SyncAgent* untuk mengirimkan *WatcherAgent* ke server agar dapat melakukan sinkronisasi *file.*
3.	*User*	*Stop SyncAgent*	Pengguna menghentikan sinkronisasi *file* dengan cara membuang *(dispose) WatcherAgent.*
4.	-	*Synchronizing files*	Sinkronisasi *file* berjalan pada server. Sinkronisasi dijalankan oleh *WatcherAgent.*
5.	*Web User*	*Upload Files*	Pengguna Web melakukan pengunggahan *file* pada server.

Use Case diagram dari aplikasi ini ditunjukkan pada Gambar 4 sedangkan deskripsi *use case* dan aktor-aktor yang terlibat didalamnya, dapat dilihat pada Tabel 1.

3. Hasil dan Analisa

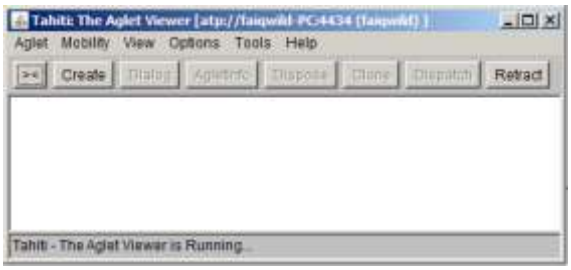
3.1. Pengujian Program

Pengujian ini dilakukan untuk mengetahui cara penggunaan program, kecepatan program dan seberapa besar kemampuan program untuk melakukan sinkronisasi. Pengujian program ini dilakukan dengan menggunakan dua *host* dimana kedua *host* akan menjalankan sinkronisasi. Selain melakukan sinkronisasi, satu dari dua *host* tersebut akan bertindak sebagai *user* yang menjalankan *SyncAgent*.

3.1.1. Penggunaan Program

Pada penggunaan program akan ditunjukkan bagaimana cara menjalankan *SyncAgent* dari awal sampai akhir. Selain itu juga ditunjukkan bagaimana menampilkan kolom Backup Information.

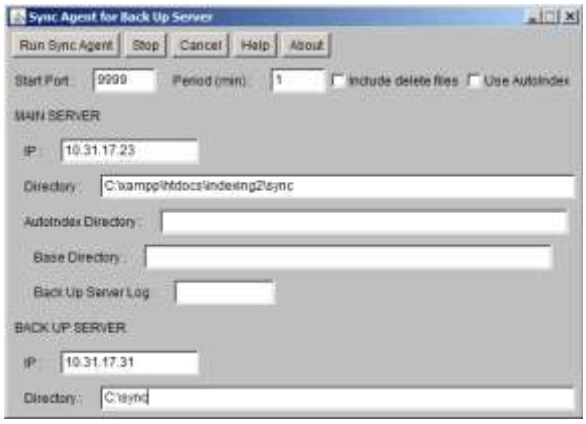
3.1.1.1. SyncAgent



Gambar 5. Tampilan Tahiti Server

Gambar 6. Create Aglet

Pertama yang perlu dilakukan untuk menjalankan *SyncAgent* adalah menjalankan Aglets 2.02 di kedua *host.* Tampilan utama dari Aglets adalah Tahiti Server. Tahiti Server ditunjukkan pada Gambar 5. Kemudian membuat (*create*) *SyncAgent* dengan cara klik tombol "Create" pada Tahiti. Kemudian memilih *SyncAgent* pada *window* Create Agent. Lalu klik tombol "Create". Gambar 6 menunjukkan tampilan *window* Create Agent.



Gambar 7. GUI *SyncAgent*

Setelah *SyncAgent* sudah dibuat maka akan muncul tampilan yang ditunjukkan pada Gambar 7. Informasi dari host pertama diisikan pada form MAIN SERVER, sedangkan informasi dari host kedua pada form BACKUP SERVER. Form pada AutoIndex Directory, Base Directory dan Backup Server Log diisi sama dengan pengaturan pada AutoIndex PHP Script apabila Use AutoIndex diaktifkan. Setelah semua form diisi, klik tombol "Run Sync Agent" untuk menjalankan *SyncAgent* dan klik tombol "Stop" untuk menghentikan kerja *SyncAgent.*

3.1.1.2. AutoIndex PHP Script

Pertama, Masuk ke halaman web dari AutoIndex PHP Script. Kemudian login menggunakan akun Admin. Pada menu admin klik "Reconfigure Script" kemudian isi form pada "base_dir" dan "backup_log". "base_dir" menunjukkan letak direktori dasar yang akan ditampilkan sedangkan "backup_log" untuk menampilkan kolom Backup Information. Tampilan pengaturan kolom Backup Information dapat dilihat pada Gambar 8. Tampilan kolom Backup Information dapat dilihat pada Gambar 9.

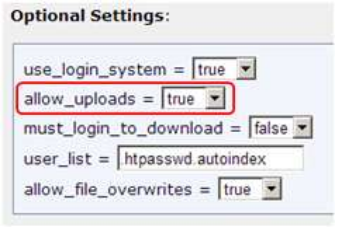
Gambar 8. Pengaturan "base_dir" dan "backup_log"

Gambar 9. Tampilan Backup Information

Setiap informasi yang ditampilkan pada kolom Backup Information berupa tulisan *last modified file* dengan ketentuan sebagai berikut :

1. Berwarna abu-abu bertuliskan "*not synced*", artinya *file* belum pernah disinkronkan.
2. Berwarna hijau, artinya *file* sudah disinkronkan.
3. Berwarna merah, artinya *file* lebih baru (*update*) belum disinkronkan.

3.1.2. Penambahan, Pembaruan dan Penghapusan *File*



Gambar 10. Pengaturan pengunggahan bagi *user*

Ada dua cara untuk menambahkan *file* dan memperbarui *file* menggunakan *SyncAgent*. Pertama, salin dan tempel (*copy paste*) *file* ke direktori target. Ini bisa dilakukan secara langsung atau melalui *remote*. Kedua, mengunggah *file* melalui form *upload* pada halaman Admin. Cara kedua merupakan cara yang lebih mudah, karena *user* tidak perlu masuk ke dalam sistem hanya perlu mengakses melalui web. *User* yang melakukan pengunggahan tidak harus menggunakan akun Admin karena Autoindex PHP Script memberikan pengaturan pengunggahan selain Admin. Gambar 10 menunjukkan letak pengaturan pengunggahan bagi user. Kemudian cara untuk menghapus *file* hanya dapat dilakukan secara langsung maupun melalui *remote*

3.1.3. Pengujian Sinkronisasi

Pengujian sinkronisasi bertujuan untuk mengetahui waktu yang dibutuhkan *SyncAgent* menjalankan sinkronisasi *file* dalam satu kali siklus. Pengujian dilakukan dengan memberikan variasi ukuran *file* dan jumlah *file*. Koneksi antara kedua host menggunakan kabel UTP *cross* secara *point-to-point*. Kondisi dan kebutuhan pengujian sinkronisasi dapat dilihat pada Tabel 2.

Tabel 2. Kondisi dan kebutuhan pengujian sinkronisasi

Host	Server	Alamat IP	Spesifikasi Host	
			Perangkat Keras	Perangkat Lunak
Host 1 (User)	Main Server	10.31.17.23	• Intel Core i3 2.13 GHz • Harddisk 300 GB • RAM 4 GB • Realtek PCIe GBE Family Controller	• Windows 7 • JAVA 7 • Aglets 2.02
Host 2	Backup Server	10.31.16.31	• AMD C-50 1.00 GHZ • Harddisk 300 GB • RAM 2 GB • Atheros AR8152/8158 PCI-E Fast Ethernet	• Windows 7 • JAVA 7 • Aglets 2.02

3.1.3.1. Pengujian dengan Variasi Ukuran File

Tabel 3. Pengujian Sinkronisasi dengan variasi besar *file*.

No.	Besar *file* (MB)	Waktu Sinkronisasi *WatcherAgent* (ms)		Waktu rata-rata sinkronisasi (ms)	Kecepatan rata-rata (MB/s)
		1	2		
1.	9,33	3654	3692	3673	2,540158
2.	18,6	5604	5758	5681	3,274071
3.	28	6786	6620	6703	4,177234
4.	37,3	8877	8727	8802	4,237673
5.	46,6	11764	11726	11745	3,967646
6.	56	12980	12932	12956	4,322322
7.	65,3	16539	16363	16451	3,969364
8.	74,7	16770	16576	16673	4,480297
9.	84	18611	18454	18532,5	4,532578
10.	93,3	21701	21615	21658	4,307877
11.	102	22729	22539	22634	4,506495
12	112	24691	24866	24778,5	4,520048
13	121	26801	26647	26724	4,527765
14	130	28600	28782	28691	4,531038

Pengujian ini dilakukan dengan cara memberikan *file* baru secara berkala dengan kelipatan 9,33 MB. Kemudian mencatat waktu yang dibutuhkan program untuk menyelesaikan sinkronisasi pada kedua *WatcherAgent*. Hasil pengujian dapat dilihat pada Tabel 3.

Berdasarkan Tabel 3 dapat dilihat bahwa semakin besar *file* maka semakin besar pula waktu yang dibutuhkan untuk melakukan sinkronisasi. Selain itu didapatkan kecepatan sinkronisasi *file* dengan kecepatan minimum sekitar 2,54 MB/s ,kecepatan maksimum sekitar 4,53 MB/s dan kecepatan rata-rata sekitar 4,13 MB/s. Kecepatan sinkronisasi yang tidak tetap tersebut terjadi karena aplikasi dibuat menggunakan koneksi *stream* menggunakan protokol TCP. Selama proses pengiriman *file* dimungkinkan terjadi paket data hilang, rusak ataupun kesalahan pengiriman sehingga paket data yang hilang tersebut perlu dikirimkan ulang. Ini merupakan karakteristik dari protokol TCP. Selain itu waktu rata-rata yang hampir sama dapat dilihat pada data nomor 6 dan 7. Ini disebabkan karena sinkronisasi dilakukan pada jaringan Internet aktif, sehingga waktu yang dibutuhkan untuk melakukan sinkronisasi dimungkinkan bertambah atau berkurang sesuai arus data yang melewati jaringan.

3.1.3.2. Pengujian dengan Variasi Jumlah File

Pengujian ini dilakukan dengan cara memberikan *file* baru dengan jumlah tertentu dengan ukuran yang sama yaitu 9,33 MB. Kemudian mencatat waktu yang dibutuhkan program untuk menyelesaikan sinkronisasi pada kedua *WatcherAgent*. Hasil pengujian dapat dilihat pada Tabel 4.

Tabel 4. Pengujian Sinkronisasi dengan variasi jumlah *file*.

No.	Jumlah *file* (1 file = 9,33 MB)	Waktu Sinkronisasi *WatcherAgent* (ms)		Waktu rata-rata sinkronisasi (ms)	Kecepatan rata-rata (MB/s)
		1	2		
1.	1	2459	2637	2548	3,661695
2.	2	3089	2810	2949,5	6,306154
3.	3	4416	4443	4429,5	6,321255
4.	4	4368	4447	4407,5	8,462847
5.	5	7632	7406	7519	6,197633
6.	6	6240	6012	6126	9,141365
7.	7	7044	7278	7161	9,118838
8.	8	8143	7897	8020	9,314214
9.	9	8659	8864	8761,5	9,587399
10.	10	12331	12839	12585	7,413588
11.	11	10993	10920	10956,5	9,309542
12	12	11481	11263	11372	9,848751

Berdasarkan Tabel 4 dapat dilihat bahwa semakin banyak jumlah *file* (dengan ukuran masing-masing sama) semakin besar pula waktu yang dibutuhkan untuk melakukan sinkronisasi. Ini terbukti dari nilai waktu rata-rata sinkronisasi yang cenderung bertambah dari 2548 ms sampai 11372 ms. Namun dari hasil perhitungan kecepatan sinkronisasi didapatkan semakin banyak jumlah *file* maka kecepatan sinkronisasi cenderung naik. Ini dikarenakan proses sinkronisasi dijalankan secara *multithreading.* Setiap *file* disinkronkan oleh satu *thread.* Semakin banyak *file* semakin banyak *thread* dibuat dan *thread-thread* tersebut dieksekusi secara bersamaan. Meski demikian, penurunan kecepatan sinkronisasi terjadi pada data nomor 5 dan 10. Ini dikarenakan lalu lintas data pada jaringan mempengaruhi *latency* pada koneksi *stream* masing-masing *thread. Latency* merupakan waktu yang dibutuhkan untuk menjalin koneksi antara kedua *host.* Apabila arus data pada jaringan meningkat maka semakin lama waktu yang dibutuhkan untuk menjalin koneksi antara kedua *host* dan ini berakibat pula pada waktu rata-rata sinkronisasi yang semakin lama.

3.1.3.3. Pengujian dengan Variasi Ukuran dan Jumlah File

Pengujian ini dilakukan dengan cara memberikan *file* baru dengan jumlah tertentu dengan ukuran yang berbeda-beda. Ukuran tersebut dibuat secara acak menggunakan program dengan range antara 1 byte sampai dengan 20 MB. Kemudian mencatat waktu yang dibutuhkan program untuk menyelesaikan sinkronisasi pada kedua *WatcherAgent*. Hasil pengujian dapat dilihat pada Tabel 5.

Tabel 5. Pengujian Sinkronisasi dengan variasi ukuran dan jumlah *file*.

No.	Jumlah *file*	Ukuran total *file* (MB)	Waktu Sinkronisasi WatcherAgent (ms)		Waktu rata-rata sinkronisasi (ms)	Kecepatan rata-rata (MB/s)	Keterangan
			1	2			
1.	100	213	29506	29475	29490,5	7,222665	Berhasil
2.	200	455	53711	53661	53686	8,475208	Berhasil
3.	300	694	81307	81260	81283,5	8,538018	Berhasil
4.	400	915	118628	118670	118649	7,711822	Berhasil
5.	500	1146,88	136624	136781	136702,5	8,389605	Berhasil
6.	600	1392,64	154975	155017	154996	8,985006	Berhasil
7.	700	1669,12	198744	198900	198822	8,395047	Berhasil
8.	800	1935,36	224731	224796	224763,5	8,610651	Berhasil
9.	900	2170,88	254974	255014	254994	8,513455	Berhasil
10.	1000	2426,88	279194	279240	279217	8,691734	Berhasil
11.	1100	2662,4	330409	330457	330433	8,057307	Berhasil
12	1200	2918,4	339036	339877	339456,5	8,597272	Berhasil

Berdasarkan pengujian pada Tabel 5 dapat dijelaskan sebagai berikut. Sinkronisasi *file* menggunakan data sampai dengan 1200 *file* berhasil dilakukan. Semakin banyak *file* dengan ukuran bervariasi maka waktu yang dibutuhkan untuk melakukan sinkronisasi cenderung meningkat. Selain itu didapatkan kecepatan yang bervariasi pula dengan kecepatan minimum sekitar 7,22 MB/s, kecepatan maksimum sekitar 8,98 MB/s dan kecepatan rata-rata sinkronisasi *file* adalah 8,34 MB/s. Namun pada data nomor 1 dan 4 terjadi penurunan sekitar 1 MB/s terhadap data lainnya. Hal ini dikarenakan lalu lintas data pada jaringan yang terkadang tinggi memperlambat komunikasi yang dibuat secara *multithreading. Thread* maksimal yang dibuat pada aplikasi ini jumlahnya adalah 25. Setiap mencapai 25 *thread* maka proses sinkronisasi berhenti pada selang waktu tertentu untuk menunggu semua *thread* selesai berjalan. Setelah selesai, satu *host* memberitahukan kepada *host* lain bahwa sudah siap melanjutkan sinkronisasi. Namun proses ini terpengaruh oleh lalu lintas data pada jaringan, dan juga *latency* tiap *thread*, sehingga terkadang terjadi penurunan kecepatan sinkronisasi.

4. Kesimpulan

Sinkronisasi pada penambahan dan pembaruan *file* berjalan normal. Sinkronisasi pada mode *Include delete files* terkadang gagal. Hal ini dikarenakan tepat setelah *file* dihapus, *WatcherAgent* menerima *file* yang sama dengan nama *file* yang dihapus. Kecepatan rata-rata sinkronisasi *file* adalah 8,34 MB/s. Selain disebabkan oleh karakteristik dari komunikasi TCP sendiri, penurunan kecepatan sinkronisasi juga disebabkan oleh *latency* pada jaringan karena lalu lintas data pada jaringan terkadang meningkat. Sinkronisasi dilakukan menggunakan *multithreading* maka masing-masing *thread* dimungkinkan terjadi *latency* yang berbeda-beda, sehingga kecepatan sinkronisasi dapat berubah-ubah. Auto Index PHP Script dapat menampilkan informasi sinkronisasi *file* pada kolom Backup Information, namun karena tidak ada (fitur) batasan jumlah *file* yang ditampilkan pada satu halaman, Auto Index PHP Script hanya dapat menampilkan 1400 *file* dan atau folder dalam satu halaman. Pengunggahan *file* menggunakan Auto Index PHP Script berjalan normal. Sebaiknya dua server diletakkan pada satu jaringan untuk mendapatkan kecepatan sinkronisasi yang maksimal. Pada mode *Include delete files* disarankan menghapus *file* ketika sinkronisasi tidak sedang berlangsung untuk menghindari kegagalan sinkronisasi. Pengembangan fitur batasan jumlah *file* dalam satu halaman pada Auto Index PHP Script perlu dilakukan untuk mendapatkan hasil tampilan yang lebih baik. Sinkronisasi *file* akan lebih efisien apabila menggunakan mekanisme *checksum* dan menggunakan port sesedikit mungkin. Sinkronisasi *file* akan lebih bermanfaat apabila ditambahkan fitur pelaporan sinkronisasi via email ataupun SMS *gateway*. Perlu dilakukan penelitian lebih lanjut tentang keamanan data yang dibawa oleh *agent*. Perlu dilakukan penelitian lebih lanjut untuk mengetahui bagaimana *agent* dapat dibuat dan tidak bergantung pada Tahiti Server.

Referensi

[1]. Naylor, M., "The Use of Mobile Agents in Network Management", MSc Information Technology (Systems Integration), 2000.

[2]. P. Dasgupta, N. Narasimhan, L.E. Moser and P.M. Melliar-Smith, "MAgNET: Mobile Agents for Networked Electronic Trading" *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, Vol. 11, No. 4, 1999.

[3]. Purbasari, Intan Yuniar and, Joko Lianto, "Rancang Bangun Perangkat Lunak Agent untuk Polling Halaman Web pada Selancar Web dalam Komunitas Menggunakan Aglet". *Jurnal Teknologi Informasi dan Komunikasi*, 2 (1). pp. 15-21. ISSN 1978-0087, 2006.

[4]. Yudianto, Scifo Anggi, Somantri, M., dan Isnanto, R. Rizal, "Perancangan dan Pembuatan Perangkat Lunak Berbasis Mobile Agent untuk Pencarian Buku", Universitas Diponegoro, Semarang, 2011.