SISTEM OTENTIKASI SINGLE SIGN-ON MENGGUNAKAN ALGORITMA DIFFIE-HELLMAN DAN MENGGUNAKAN DATABASE PARALLEL DENGAN MENGGUNAKAN RMI *(REMOTE METHOD INVOCATION)*

Ignatius Wijaya Kusuma[1], Puspaningtyas Sanjoyo Adi[2]
[1]Mahasiswa Teknik Informatika Universitas Sanata Dharma, Yogyakarta, Indonesia
E-mail : azterix2far4u@gmail.com

[2]Teknik Informatika Universitas Sanata Dharma, Yogyakarta, Indonesia
E-mail : puspa@usd.ac.id

ABSTRAK

Otentikasi adalah suatu tindakan untuk mengkonfirmasikan suatu kebenaran dari sebuah entitas dan sebuah trusted third-party dibutuhkan ketika sebuah transaksi saling membutuhkan kepercayaan antara orang yang bertransaksi dan orang yang ditransaksi. Penelitian ini mengembangkan suatu sistem otentikasi dengan menggunakan pendekatan sistem terdistribusi dengan menggunakan paralel database dan alogiritma Diffie-Hellman Ide utama dari parallel DBMS(Database Manajement System) adalah untuk membangun sebuah computer yang sangat kuat dari banyak komputer kecil, yang masing-masing memiliki rasio kinerja dan harga yang sangat baik dengan rasio biaya yang jauh lebih rendah bila dibandingkan dengan sebuah komputer mainframe. Melalui bantuan metode pemanggilan objek, teknologinya akan digunakan untuk memanggil fungsi dan prosedur secara remote/jarak jauh. Salah satu teknologinya adalah RMI(Remote Method Invocation). Hasil utama dari sistem ini berupa sebuah library java yang digunakan sebagai fungsi login pada sistem developer.Hasil penelitian ini menunjukkan bahwa sistem yang dikembangkan mampu mengotentikasi seorang user dalam waktu 0,125 detik. Sistem juga tetap berjalan walau ada salah sat mesin database tidak aktif.

Kata Kunci : *Otentikasi, trusted third-party, Diffie-Hellman key exchange, parallel DBMS(Database Manajemen System), RMI(Remote Method Invocation)*

1. Pendahuluan

Banyak program aplikasi sekarang yang menggunakan otentikasi contohnya: jejaring sosial seperti facebook, plurk, google mail, dan sebagainya. Tentunya banyak sekali data otentikasi yang dibutuhkan (*username* dan *password)*/*account*. Dengan pendekatan *single sign-on* atau penggunaan orang ke-tiga sebagai solusi yang dapat digunakan untuk mengatasi hal tersebut. Sistem ini dibuat pada sektor pembuatan aplikasi, dimana seorang developer aplikasi desktop maupun web tidak perlu membuat otentikasi tersendiri. Sistem yang akan digunakan oleh developer adalah berupa sebuah *library* yang secara keseluruhan mengotentikasi dan mengatur sistem bagi pengguna nantinya.

Selain daripada pengaturan otentikasi yang terpusat, dibutuhkan sebuah pengamanan khusus untuk menjaga agar data dapat disembunyikan sedemikian rupa, agar data yang dimiliki oleh pemilik *account* hanya dapat diakses oleh dan hanya oleh pemilik *account* itu sendiri. Salah satunya adalah dengan menggunakan algoritma *Diffie-Hellman Key Exchange Protocol*.

Sistem otentikasi yang dikembangkan menggunakan sistem orang ke tiga(*third-person authentication)*. Seorang *user* menggunakan jasa orang ke tiga yang dimana orang ke tiga disini sebagai seorang yang dipercaya*(trusted),* sehingga proses pengiriman data diatur oleh orang ke tiga tersebut. Hal ini dikarenakan pengecekan antara pengirim dan penerima juga diperhatikan sungguh-sungguh dan selain itu dikarenakan adakalanya dimana seorang pengirim/penerima bukan benar-benar sebagai pengirim/penerima itu sendiri. Oleh karena itu, fungsi orang ke tiga disini sebagai saksi bahwa seorang pengirim/penerima adalah orang yang benar-benar patut menerima/mengirim sebuah data. Sistem yang akan dibuat berupa *java class library, server, server database, dan web server* dimana masing-masing memiliki fungsi tersendiri dan terhubung satu sama lainnya dengan bantuan *RMI*(*Remote Method Invocation*).

2. Dasar Teori

2.1. Otentikasi

Otentikasi berasal dari bahasa Yunani dan terdiri dari dua kata, yaitu *αυθεντικός* yang artinya nyata atau asli dan *authentes* yang artinya penulis. Otentikasi adalah suatu tindakan untuk mengkonfirmasikan suatu kebenaran dari sebuah entitas. Semua ini melibatkan konfirmasi identitas seseorang, menelusuri asal-usul artefak, memastikan bahwa sebuah produk sesuai dengan apa yang di *klaim*, atau menjami bahwa sebuah program computer dapat dikatakan terpercaya*(trusted).*

Fungsi otentikasi yang dibangun pada sistem yaitu sebagai sarana untuk membangun sebuah sistem *database* yang bersifat parallel *(parallel DBMS)*, dimana proses otentikasi digunakan untuk melakukan pengecekan data yang telah diisi terlebih dahulu.

2.2. *Single Sign-On*

Menurut *The Open Group*[7], sebuah IT sistem yang mendukung proses bisnis, pengguna dan sistem *administrator* dihadapkan dengan bertambah banyaknya dan rumitnya sebuah antarmuka untuk memenuhi pekerjaan mereka masing-masing. Seorang pengguna biasanya harus *sign-on* pada beberapa sistem, yang juga memerlukan dialog *sign-on* yang sama, dan mungkin tiap sistem tersebut menggunakan pengguna yang berbeda dan juga informasi otentikasi yang berbeda. Hal ini membuat sistem *administrator* dihadapkan bagaimana memanajemen *account* pengguna di tiap-tiap sistem yang dapat diakses secara ter-koordinasi dalam rangka menjaga integritas serta keamanannya.

2.3. *Trusted Third Party*

Menurut [8], *trusted third-party* disini adalah sebuah cara penanganan dalam bagaimana cara bertransaksi dengan menggunakan pihak ketiga, dimana pihak ketiga disini dinyatakan *trusted* oleh masing-masing pihak yang bertransaksi. Sebuah *trusted third-party* dibutuhkan ketika sebuah transaksi saling membutuhkan kepercayaan antara orang yang bertransaksi dan orang yang ditransaksi. Dengan bantuan pihak ketiga, masing-masing pihak memberikan suatu kata kunci yang masing-masing dimiliki oleh kedua belah pihak, kemudian pihak ke tiga memeriksa apakah kata kunci tersebut benar atau tidak. *Trusted third-party* disini berfungsi sebagai pem-*verify* sebuah *integrity* dari pihak yang bersangkutan.

2.4. *Diffie Hellman Key Exchange*

Diffie-Hellman key exchange adalah sebuah fungsi spesifik dalam pertukaran kunci. Ini adalah salah satu contoh praktis paling awal dari sebuah fungsi pertukaran kunci yang diimplementasikan dalam bidang kriptografi. *Diffie-Hellman key exchange* memungkinkan kedua pihak yang tidak memiliki pengetahuan satu sama lain untuk bergabung membuat sebuah kunci rahasia bersama melalui jaringan komunikasi yang tidak aman. Kunci ini kmudian dapat digunakan untuk mengenkripsi komunikasi berikutnya menggunakan *symmetric key chiper*.

Gambar 1. Diffie-Hellman Key Exchange Protocol

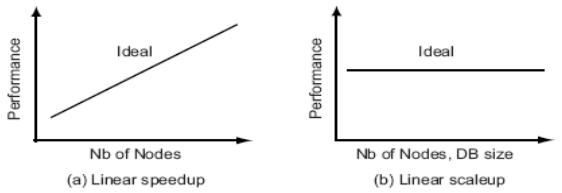
Dengan kriptografi *symmetric* sangat diperlukan adanya pengiriman kunci rahasia untuk kedua belah pihak yang berkomunikasi sebelum komunikasi yang aman dapat terbentuk. Sebelum munculnya kriptografi yang menggunakan kunci publik, pembentukan kunci rahasia bersama antar pihak yang berkomunikasi selalu menjadi masalah yang sulit, sebab pekerjaan/tugas yang dilakukan membutuhkan saluran/*channel* yang benar-benar aman, dan sering dikatakan bahwa saluran tersebut berarti penyerahan fisik kunci oleh kurir khusus. Keuntungan paling penting dari kriptografi *public key* yang menyediakan *symmetric public key* adalah pencapaian pertukaran kunci rahasia antara pihak komunikasi jarak jauh tanpa membutuhkan suatu saluran yang aman.

2.5. *DBMS (Database Management System)*

Sistem *database* dapat mengeksploitasi paralelisme dalam pengelolaan data dalam rangka untuk memberikan *database database* kinerja tinggi dan kemampuan ketersediaan data yang tinggi. Dengan demikian, mereka dapat mendukung *database* yang sangat besar dengan beban yang sangat tinggi. Idealnya, sebuah *parallel* DBMS harus mampu memberikan hal-hal berikut:

- *High Performance:* hal ini dapat diperoleh dengan beberapa solusi, yaitu: *database-oriented operating system support*, manajemen data paralel, optimisasi *query*, dan *load balancing*.
- *High-availability:* karena *parallel* DBMS terdiri dari banyak data komponen yang sama(redudant) sehingga dapat meningkatkan baik ketersediaan data maupun toleransi kesalahan data.
- *Extensibility:* adalah kemampuan untuk memperluas sistem dengan menambahkan kemampuan pengolahan dan kekuatan penyimpanan untuk sistem. Idealnya, sistem yang paralel ini harus mampu menunjukkan dua kemampuan

extensibility, yaitu: *linear speedup* dan *linear scaleup*. *Linear speedup* mengacu kepada penambahan performa/kinerja sesuai dengan penambahan jumlah node. Sedangkan *linear scaleup* mengacu kepada ketetapan performa yang dimiliki baik itu dari ukuran *database* serta jumlah node yang dimiliki.



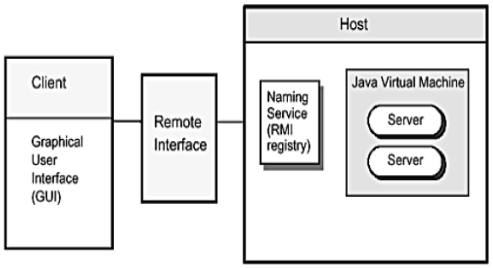
Gambar 2. Rasio Extensibility, baik dalam bentuk speedup(a) maupun scaleup(b)

2.6. *RMI (Remote Method Invocation)*

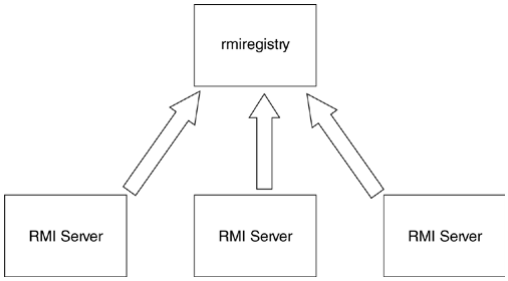
Menurut buku [1], RMI menambah pemrograman Java kekuatan dan fleksibilitas dari *remote procedure calls(RPC)*, yang juga "berorientasi objek". Hal ini menyediakan framework di dalam objek java di dalam *Java Virtual Machine(JVM)* dapat berinteraksi antar JVM. Metode pemanggilan objek bukanlah sebuah konsep yang baru. Bahkan sebelum pemrograman berorientasi objek, tekno*login*ya sudah ada dimana digunakan untuk memanggil fungsi dan prosedur secara *remote*/jarak jauh. Sistem ini seperti *remote procedure calls (RPCs)* telah digunakan selama bertahun-tahun dan terus digunakan sampai saat ini. *Remote prosedure calls* dirancang sebagai cara *platform-neutral* berkomunikasi antar aplikasi, terlepas dari sistem operasi atau perbedaan bahasa pemrograman.

Sistem yang menggunakan RMI untuk berkomunikasi biasanya dibagi menjadi dua kategori: *client* dan *database*. Sebuah *database* menyediakan layanan RMI, dan *client* memanggil metode objek dari layanan ini. Sebuah *database* RMI harus mendaftarkan *database* tersebut dengan layanan pencarian*(lookup service)*, yang mengizinkan sebuah *client* untuk menemukan mereka, atau mereka dapat membuat sebuah referensi ke layanan tersebut dengan cara yang lain.

Tergabung dalam *Java Platform* adalah sebuah aplikasi bernama *rmiregistry*, dimana *rmiregistry* ini berjalan secara terpisah dan memungkinan sebuah aplikasi untuk mendaftarkan layanan RMI atau mendapatkan sebuah referensi ke sebuah layanan khusus. Setelah *database* telah terdaftar, *database* akan menunggu permintaan dari *client* RMI.



Gambar 3. Arsitektur RMI, hubungan antara client dan host



Gambar 4. menggambarkan mendaftarkan layanan dengan RMI registri tunggal

Client RMI akan mengirim RMI sebuah pesan untuk memanggil sebuah fungsi objek secara *remote*. Sebelum pemanggilan fungsi secara *remote*, *client* harus memiliki referensi *remote object*. Hal ini biasanya diperoleh dengan mencari sebuah layanan di registri RMI. Setelah sebuah referensi objek diperoleh (baik melalui *rmiregistry*, sebuah layanan pencari, atau dengan membaca URL referensi objek dari file), *client* dapat berinteraksi dengan *remote service*. Rincian jaringan dari sebuah permintaan benar-benar transparan bagi para pengembang aplikasi, dimana bekerja secara *remote object* menjadi semudah bekerja dengan sistem lokal. Hal ini dicapai melalui pemikiran cerdas dari divisi RMI yang membagi sistem RMI menjadi dua komponen, yaitu *stub* dan *skeleton*.

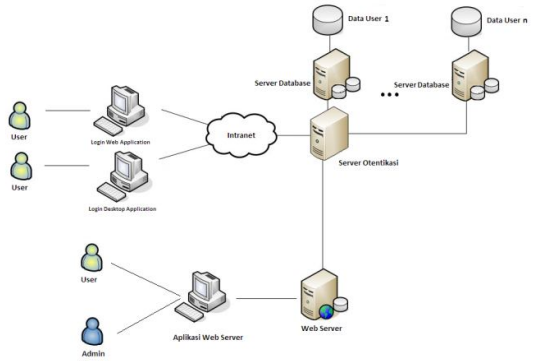
Stub disini berfungsi sebagai sebuah proxy, menyampaikan permintaan objek ke RMI *database*. Layanan RMI didefinisikan sebagai sebuah *interface*, bukan sebagai suatu implementasi. *Stub* mengimplementasikan *inteface* tertentu, dimana aplikasi *client* dapat menggunakannya sama seperti implementasi objek yang lainnya. Namun, daripada melakukan proses itu sendiri, *stub* mengirimkan pesannya itu ke penyedia layanan RMInya, menunggu respon, dan mengembalikan hasil respon tesebut kepada pengirimnnya.

Gambar 5. cara kerja stub dan skeleton

Skeleton bertanggung jawab untuk mendengarkan permintaan yang masuk dan memberikannya pada penyedia layanan RMI. *Skeleton* tidak menyediakan implementasi layanan RMI. Namun hanya bertindak sebagai penerima permintaan. Setelah pengembang membuat RMI *interface*, pengembang masih harus menyediakan implementasi yang nyata dari *interface*-nya. Implementasi ini akan dipanggil oleh *skeleton* yang fungsinya dilakukan secara tepat dan hasilnya dikirimkan kembali kepada *stub* di RMI *client*. Model ini membuat pemrograman lebih sederhana.

3. Implementasi dan Pembuatan System

Sesuai dengan uraian pada latar belakang, penggunaan *account* sangat dibutuhkan sebagai tanda pengenal pada suatu aplikasi, terutama aplikasi yang ada di *internet*. Sistem manajemen ini dirancang dengan menggunakan teknologi *Remote Method Invocation* (RMI), menggunakan bahasa pemrograman Java sehingga menjadi sebuah sistem yang menggunakan prinsip kerja sistem *Client-Server*. Arsitektur yang dikembangkan sesuai dengan gambar 6.



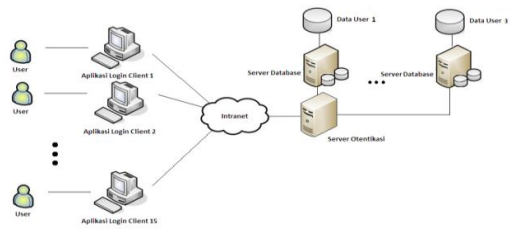
Gambar 6. Arsitektur sistem yang akan dibuat

Sistem yang dibangun ini diharapkan dapat memenuhi permintaan pengguna dalam memproses/memiliki sebuah *account* yang dapat digunakan dalam beberapa aplikasi tersebut. Dengan catatan bahwa aplikasi tersebut menggunakan sistem manajemen ini serta terkoneksi dengan *database* yang menyediakan fasilitas ini.

Sistem manajemen *account* ini terdiri dari lima aplikasi, yaitu: *web database, client* dan *database* otentikasi, *client* dan *database* kontrol *database*. *Web database* digunakan sebagai sarana untuk mendaftarkan *account* pada sistem *database* dan juga digunakan sebagai sarana bagi perancang aplikasi dalam merancang aplikasinya tanpa harus membuat sistem otentikasi secara khusus.

4. Pengujian

Pengujian pertama yaitu mengenai sisi performansi atau mampu tidaknya masing-masing client melakukan login secara bersamaan. Arsitektur pengujian dapat dilihat pada gambar 7.



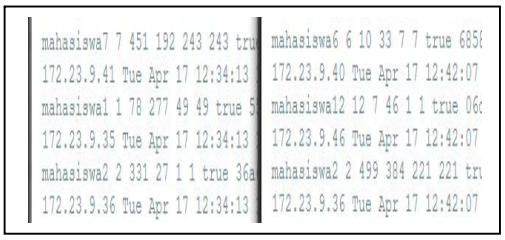
Gambar 7. Arsitektur pengujian login bersamaan

 ISBN 979 - 26 - 0255 - 0

Tabel 1. Data hasil pengujian login bersamaan

Jumlah User	Keberhasilan Login	Jumlah Login / detik
5	100%	5 per detik
10	100%	7 per detik
15	100%	8 per detik

Hasil Log :



Gambar 8 Hasil Percobaan Client Melakukan Otentikasi bersamaan

Pengujian kedua adalah mengenai performansi maksimal yang dapat digunakan:
Jumlah komputer : 8 Unit untuk keperluan

1. 3 Unit PC untuk Database Server
2. 1 Unit PC untuk Client Server
3. 4 Unit PC untuk Client

Spesifikasi per-Unit :

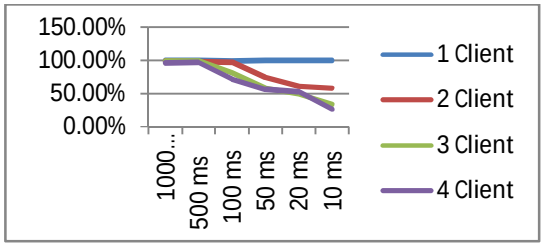
- Microsoft Windows XP Professional (5.1, Build 2600)
- System Manufactur : Gigabyte Technology Co., Ltd
- System Model : H55M-S2V
- Bios: Award Modular BIOS v6.00PG
- Processor: Intel® Core ™ i3 CPU 530 @ 2.93GHz (4 CPUs)
- Memory: 1848 RAM
- Page File: 539 used, 3202MB available
- VGA: Intel® HD Graphics
- Approx. Total Memory: 1024 MB
- LAN Speed: 100 MBps

Data yang digunakan sebagai pembanding adalah 400 *username* dan *password* yang dibagi menjadi 100 data *login* tiap aplikasi penguji. Percobaan untuk penggunaan 1 - 4 *client* dengan 1 *controller*.

Tabel 2. Hasil pengujian client dan server

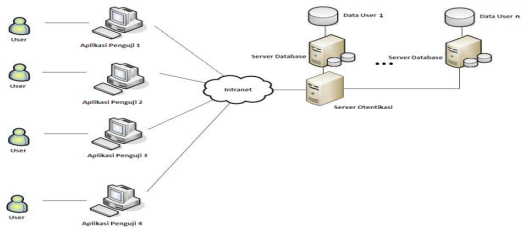
Jumlah client	Jumlah Dalam Delay / ms					
	1000	500	100	50	20	10
1	99.5 %	99.5 %	99 %	100 %	99.5 %	100 %
2	99 %	98.5 %	97.5 %	75%	61.5 %	58.5%
3	98.6 %	99 %	81.3 %	58.6 %	49.3 %	34 %
4	96 %	97.25 %	71.75 %	57 %	52.75 %	26.75 %

Perbandingan tiap *Client* untuk tiap keberhasilan dalam persentase:



Gambar 9. Grafik hasil pengujian client dan server

Pengukuran kecepatan waktu maksimal server menanggapi permintaan *client*, dengan melakukan penghitungan terhadap hasil keberhasilan berdasarkan jumlah *client* yang terbanyak dengan keberhasilan diatas 90%. Yaitu penghitungan dengan penggunaan data log 2 client dengan kecepatan penerimaan data 100 *ms*. Arsitektur sistem pengujian dapat dilihat pada gambar 10.



Gambar 10. Pengujian performa sistem

Dari data log ditemukan data akses terbanyak dalam 1 detik adalah 16 data *login*. Total data yang dikirim oleh masing-masing *client* adalah 100, sehingga total data yang dikirimkan adalah 200. Maka penghitungan data yang dikirim tiap *client* adalah 8 data *login*/detik oleh masing-masing *client*. Percobaan untuk penggunaan 2 *client* dan 2 *server*.

Jumlah Client	Jumlah dalam (Delay/ms)					
	1000	500	100	50	20	10
2	97 %	97 %	99.5 %	98 %	99 %	97 %

Percobaan untuk penggunaan 4 *client* dengan 2 *server*

Jumlah Client	Jumlah dalam (Delay/ms)					
	1000	500	100	50	20	10
4	88.75 %	98.75 %	97.25 %	87.75 %	69.75 %	55.25 %

Hasil dari percobaan testing diatas tidak jauh berbeda dengan penghitungan berdasarkan penghitungan selanjutnya, namun karena kemampuan penghitungan dilakukan dengan pencatatan yang dilakukan oleh 2 server maka kemampuan server otentikasi bertambah sehingga mampu menangani client lebih banyak dibandingkan hanya dengan 1 server saja.

5. Analisa Data

Berikut ini analisis yang berdasarkan pada aspek pengembangan sistem terdistribusi.

1. *Transparency*
   Transparansi yang ada pada sistem sinkronisasi adalah mengenai hal-hal yang diketahui dan tidak diketahui oleh *user* dalam sistem otentikasi ini. *Password* hanya diketahui oleh *user* karena sistem tidak menyimpan *password* secara biasa, melainkan terenkripsi oleh MD5. Proses kalkulasi yang dilakukan oleh sistem juga hanya diketahui oleh sistem itu sendiri sehingga keamanan dari password itu sendiri dapat dijaga. (kecuali terdapat *key logger* pada sisi client)
2. *Flexibility*
   Sistem ini dikembangakan menggunakan bahasa pemrograman java, sehingga saat dipasang pada berbagai sistem operasi seperti Windows dan Linux.
3. Reliability
   Dengan teknologi *RMI* yang menggunakan mekanisme *built-in Java security* memungkinkan sistem aman saat *user* mengambil implementasi. *RMI* menggunakan *security manager* yang ditetapkan sebagai *security*-nya.
4. *Performace*
   Dari hasil pengujian performansi yang dilakukan, sistem mampu menanggapi permintaan dari lebih dari satu *client* yang dilakukan secara bersamaan (4 client secara bersamaan dengan jumlah data tiap *client* sebanyak 100 buah), meskipun sistem mengerjakan permintaan secara berurutan (sekuensial). Hasil pembandingannya terdapat pada lampiran. Selain melakukan tes diatas, juga dilakukan login dengan jumlah 5, 10, 15 *user* secara bertahap. Berdasarkan pengujian, dalam 1 detik ada 8 proses *login* yang sukses, sehingga satu proses login diselesaikan dalam waktu 0.125 detik.

5. *Skalabilitas* (*scalability*)
   Skalabilitas sistem otentikasi ini sejauh dalam lingkup lingkungan jaringan sederhana dikarenakan pengujian sistem dilakukan pada tingkat jaringan dalam lab.

6. Kesimpulan

Setelah sistem otentikasi dibangun dan melakukan pengujian serta melakukan analisa terhadap hasil pengujian di dapatkan kesimpulan sebagai berikut :

1. Untuk setiap login pada pengujian ke dua yaitu fungsi login untuk user sebanyak 5, 10, dan 15. Didapatkan hasil yang memuaskan yaitu tingkat keberhasilan 100%. Kecepatan maksimal dari hasil tes didapatkan sebanyak 8 data *login* yang mampu dikerjakan oleh server.
2. Hasil dari uji performansi untuk kecepatan dengan data sebanyak 100 buah untuk setiap *client*-nya (client yang di ujicoba berjumlah antara satu sampai empat), menghasilkan informasi bahwa untuk mendapatkan kecepatan/kemampuan maksimal server diharuskan setiap 2 *client* dikendalikan oleh 1 *controller*. Hal ini dilakukan melihat dari rasio yang digunakan yaitu antara 90%-100% adalah sukses, dengan penghitungan total keberhasilan = jumlah *login* – jumlah *login* gagal / jumlah *client* yang digunakan. Berdasarkan pengujian, dalam 1 detik ada 8 proses *login* yang sukses, sehingga satu proses login diselesaikan dalam waktu 0.125 detik.
3. Hasil dari data yang diberikan dapat berubah tergantung dari spesifikasi komputer dan kondisi yang diberikan

Daftar Pustaka

[1] **Grosso, William.** *Java RMI.* First Edition. O'Reilly, 2001. p. 572. ISBN: 1-56592-452-5.
[2] **Ramakrishnan, Raghu and Gehrke, Johannes.** *Database Management System.* 3rd Edition.McGraw-Hill Science/Engineering/Math, 2002. p. 1098. ISBN-10: 0072465638 ISBN-13:978-0072465631.
[3] *Parallel Database Systems: The Future of High Performance Database Processing.* **DeWitt, David J. and Gray, Jim.** 1992.
[4] **Ozsu, M. Tamer and Valduriez, Patrick.** *Principles of Distributed Database Systems.* 3rd Edition. Pearson Education, Inc. p. 538. ISBN: 978-1-4419-8833-1.
[5] **Timothy A. Howes, Mark C. Smith, Gordon S. Good.** *Understanding and Deploying LDAP Directory Services.* First Edition. New Riders Publishing, 1998. p. 880. ISBN: 1-57870-070-1.
[6] **Arlow, Jim and Neustadt, Ila.** *Uml and the Unified Process: Practical Object-Oriented Analysis and Design.* 1st Edition. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN: 0201770601.
[7] **The Open Group.** *X/Open Single Sign-On Service (XSSO) - Pluggable Authentication.* UK : The Open Group, 1994. ISBN: 1859121446.
[8] **Microsoft Corporation.** *Microsoft® Exchange 2000 Server Resource Kit.* Microsoft Press, September 2000. 978-0-7356-1017-0 / 0-7356-1017-7.
[9] **Mao, Wenbo.** *Modern Cryptography: Theory and Practice.* Prentice Hall PTR, 2003. p. 648. ISBN: 0-13-066943-1.
[10] **Martin E. Hellman, Bailey W. Diffie, and Ralph C. Merkle.** *Cryptographic apparatus and method. 200,770* U.S. Patent #4, April 29, 1980. Algoritm.
[11] **Reilly, David and Reilly, Michael.** *Java Network Programming and Distributed Computing.* Addison-Wesley Longman Publishing Co., Inc., 2002. p. 496. ISBN: 0-201-71037-4.
[12] **HorstMann, Cay S. and Cornell, Garry.** *Core Java™ 2 - Advanced Features.* 7th Edition. Prentice Hall PTR, 2004. p. 1024. Vol. II. ISBN: 0-13-111-826-9.
[13] **Yadav, Subhash Chandra and Singh, Sanjay Kumar.** *An Introduction to Client/Server Computing.* New Age International (P) Ltd, Publisher, 2009. p. 200. ISBN (13): 987-81-224-2861-2.
[14] **Widiatmoko, Michael Bangkit.** *Pemanfaatan Sistem Terdistribusi Remote Method Invocation Untuk Sinkronisasi File Sebagai Sarana Backup.* Yogyakarta : Universitas Sanata Dharma, 2011. Tugas Akhir.
[15] **Pitt, Esmond and McNiff, Kathy.** *Java.rmi: The Remote Method Invocation Guide.* Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN: 0201700433.