ANALISIS PERFORMA SISTEM CLUSTER PADA PROSES DISTRIBUTED RENDERING MENGGUNAKAN OPEN SOURCE SOFTWARE

Dian Prawira

Sekolah Tinggi Manajemen Informatika dan Komputer Pontianak
Program Studi Teknik Informatika
Jln Merdeka No 372 Pontianak, Kalimantan Barat
Email: **wiradianxiv@mti.gadjahmada.edu**

Abstract : *Animation industry in Indonesia seems to be very prosperous as can be seen from the big amount of animation production houses. However, these production houses are not supported by high end computer graphics as they require to render complex animation video since these kind of computers are very costly. Therefore, computer system which is cheap can process complex animation rendering is needed. In this research, we need to setup available PCs as cluster computer system doing the distributed rendering process and analyze its performance for many kinds of rendering tasks. The method used in this research is laboratory experiments on five computer units connected one another and clustering using DrQueue as middleware. This technique will lead to resources sharing and rendering process distribution. The results shows that, the tested cluster system of distributed rendering process can perform the computation more efficiently when done the complex animation rendering. The results shows that the value of efficiency is higher when render complex animation.*

Keyword: cluster, distributed rendering, paralel computing

1. PENDAHULUAN

Teknologi komputer saat ini merupakan hal yang penting dalam menyokong perkembangan teknologi komputasi. Begitu pula dengan pada dunia animasi. Di negara-negara maju, perkembangan industri animasi tiga dimensi berkembang drastis dikarenakan prasarana komputer yang digunakan sangat memadai untuk membuat sebuah animasi yang sangat kompleks. Sebab animasi yang kompleks memerlukan *resource* komputer yang besar, sedangkan *resource* komputer yang besar memerlukan biaya yang tidak sedikit pula. Komputasi paralel merupakan salah satu alternatif pemecahan masalah kurangnya *resource* untuk melakukan proses tersebut. Inti dari komputasi paralel yaitu *hardware*, *software*, dan aplikasinya. Proses paralel merupakan suatu *pemrosesan* informasi yang mendekatkan pada manipulasi rata-rata dari elemen data terhadap satu atau lebih penyelesaian proses dari sebuah masalah(Laksono dkk, 2004). Penelitian ini membahas mengenai strategi infrastruktur dengan memanfaatkan *resource* komputer dengan spesifikasi yang rendah untuk melakukan *rendering* yang kompleks, serta penggunaan sistem *distributed rendering* ini dalam mempercepat proses *rendering* yang terjadi

2. TINJAUAN PUSTAKA

Pada penelitian yang dilakukan oleh Laksono, dkk (2004), *distributed rendering* yang dilakukan menggunakan dua jenis *middleware* yaitu Openmosix dan MPI. Pada hasil yang mereka dapatkan bahwa *distributed rendering* bekerja lebih optimal ketika menggunakan MPI dibandingkan dengan menggunakan Openmosix hal ini dikarenakan sistem *cluster* yang digunakan Openmosix akan terus bekerja selama *server* tersebut

belum sampai pada titik jenuh, dan baru akan pindah ke *client* jika sudah sampai pada titik jenuh. Sedangkan pada MPI beban kerja langsung didistribusikan secara merata pada *client* dan *server*.

Putri, dkk.(2004) pada penelitiannya menjalankan *middleware* Openmosix pada dua kondisi yang berbeda, yaitu dengan menggunakan *disk* (*hard disk drive*), dan tanpa menggunakan *disk* (*hard disk drive*). *Software* yang berjalan pada *middleware* tersebut menggunakan *software* POVRAY. Dari penelitian mereka didapat hasil bahwa tidak terdapat perbedaan kecepatan yang signifikan antara proses *rendering* menggunakan komputer *cluster* yang menggunakan *disk* dan tanpa disk, namun dari segi penghematan penggunaan *hardware*, hal ini cukup menjadi bahan pertimbangan

2.1 Cluster

Definisi *cluster* dalam dunia teknologi informasi dapat memiliki beberapa pengertian. Secara umum definisi *cluster* adalah sejumlah komputer (PC ataupun *workstation*) yang digabungkan sebagai satu kesatuan dengan bantuan piranti lunak dan jaringan komputer(Jamal & Sistha, 2006).

Kegunaan dari PC *cluster* dapat dibedakan dalam tiga jenis, yaitu: 1) untuk meningkatkan ketersediaan dari sistem yang handal (*reliable*) atau dikenal dengan *High Availability* (HA). Hal ini diperlukan untuk sistem yang menjalankan *mission-critical* yang memiliki kontinuitas dari fungsi sangat kritis; 2) *Load Balancing Clusters*, yang umumnya digunakan pada *web-server* yang sangat sibuk seperti *search engine* Google. Disini beberapa *computer node* menjadi *host* dari *website* yang sama, dan jika ada permintaan untuk mengkakses ke halaman *web* ini, maka akan diarahkan ke *computer node* yang bebannya lebih rendah; dan 3) untuk komputasi kinerja tinggi atau *High Performance Cluster* (HPC), yang merupakan tujuan dari pengguna tradisional dari PC *cluster*, seperti peneliti, industri riset dan pengembangan. Dalam hal ini *cluster* digunakan dengan cara menjalankan program secara paralel untuk aplikasi yang sangat banyak memakan waktu (*time consuming*).

Ada dua macam tipe *cluster* yang dominan, yaitu : *1) High Performance Computing (HPC):* Secara umum, tipe *cluster* HPC ditujukan pada bagaimana suatu proses komputasi diakselerasi, dengan demikian task bisa diselesaikan lebih cepat contoh *clustering* jenis ini adalah openMosix; dan *2) High Availability (HA). Secara umum,* tipe *cluster* ini ditujukan agar program yang dijadikan di atasnya bisa terus berjalan, sekalipun salah satu *node hang* atau *down*. Contoh yang paling mudah adalah *web server* Apache yang diatur dengan suatu *redirector*, sehingga jika salah satu *server down*, *server* lain bisa mengambil alih.

Setiap *node* di dalam komputer *cluster* yang dibangun di dalam penelitian ini adalah komputer *cluster* yang menggunakan sistem operasi Linux dan perangkat-perangkat *input* dan *output* yang mirip serta menggunakan konsep *High Performance Computing* (HPC).

2.2 Ukuran Kinerja Komputasi Paralel

Banyak parameter yang dapat digunakan untuk mengukur kinerja sistem paralel, diantaranya adalah *speedup* komputasi paralel, efisiensi dan waktu komputasi(Eager, 1989). Pada penelitian ini parameter-parameter yang akan diambil untuk dijadikan sebuah penilaian dari kinerja komputasi paralel menggunakan sistem *distributed rendering* ini yaitu:

1) Speedup Komputasi Paralel

Parameter yang sangat penting untuk mengukur kinerja suatu program paralel adalah waktu eksekusi dan *speedup*. Waktu eksekusi dapat diartikan sebagai waktu berlangsungnya (*running*) program paralel pada arsitektur komputer paralel yang dituju. Waktu eksekusi sekuensial didefinisikan sebagai waktu *running* algoritma yang sama yang dieksekusi oleh satu prosesor. *Speedup* dari suatu program paralel adalah waktu eksekusi sekuensial dibagi dengan waktu eksekusi paralel. *Speedup* dapat didefinisikan seperti pada persamaan berikut ini:

$$S(n)=\frac{T(1)}{T(n)} \quad \text{..... (1)}$$

*S(n) = speedup, T(1) = waktu eksekusi operasi pada sistem satu prosesor, T(n) = waktu eksekusi pada sistem n prosesor

2) Efisiensi

Dalam mengukur kinerja suatu sistem paralel efisiensi tidak dapat dipisahkan dari *speedup*. Efisiensi didefinisikan sebagai rata-rata penggunaan n prosesor yang dialokasikan untuk suatu proses komputasi paralel (Eager,1989). Dari definisi *speedup* tersebut, lahir nilai efisiensi, E(n), untuk sistem dengan n prosesor. Efisiensi didefinisikan seperti pada persamaan berikut ini:

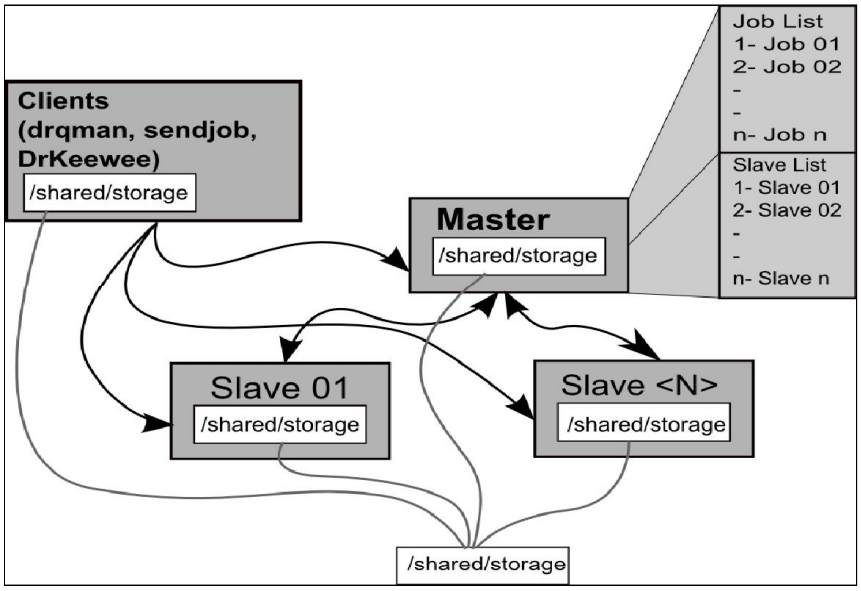
$$E(n)=\frac{S(n)}{n}=\frac{T(1)}{nT(n)} \quad \text{..... (2)}$$

**S(n) = speedup, T(1) = waktu eksekusi operasi pada sistem satu prosesor, T(n) = waktu eksekusi pada sistem n prosesor, E(n) = nilai efisiensi n prosesor

2.3 DrQueue

Untuk melakukan implementasi dari *distributed rendering* pada sistem ini maka digunakanlah middleware DrQueue yang merupakan middleware open source. Prinsip kerja DrQueue adalah membagi tugas rendering pada beberapa komputer slave yang diperintahkan oleh master (Gambar 1). Komputer client yang merupakan pengguna yang bermaksud melakukan rendering terhadap suatu file mengirimkan file yang akan dilakukan proses render pada komputer master.

Dari komputer master memecah tugas *rendering* tersebut menjadi beberapa *task* yang dipetakan pada tiap-tiap *slave*. Setelah terjadinya pemetaan, maka task tersebut ditempatkan pada *shared storage* yang akan diambil sendiri oleh komputer *slave* pada posisi *idle*, untuk dilakukan proses *rendering* pada masing-masing komputer *slave* sesuai dengan pemetaan task yang dilakukan pada komputer master. Apabila proses *rendering* yang dilakukan oleh komputer *slave* telah selesai, maka hasil *render* disimpan pada *shared storage*, dan komputer *slave* yang melakukan proses *rendering* tadi memberitahu pada komputer master bahwa proses *rendering* telah selesai dilakukan. Secara otomatis komputer master akan melakukan penggabungan hasil-hasil *rendering* menjadi satu sesuai pemetaan yang telah dilakukan sebelumnya.



Gambar 1. *Konsep dasar sistem distributed rendering pada DrQueue*

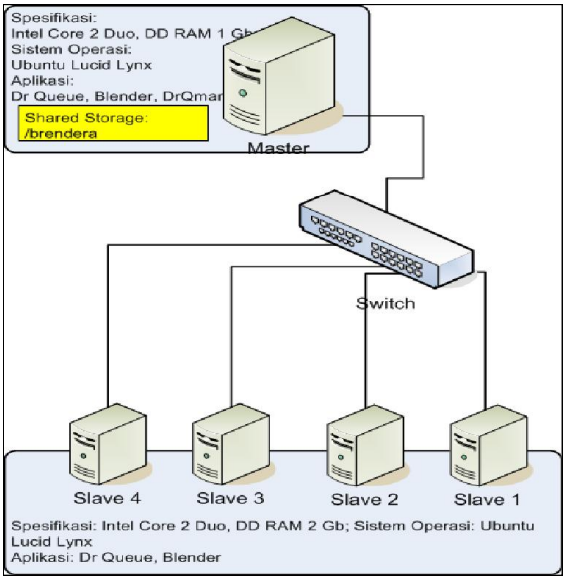
3. METODOLOGI PENELITIAN

Penelitian ini merupakan sebuah penelitian eksperimental dengan mengimplementasikan sistem *cluster* dengan *middleware* Dr. Queue. Penelitian ini menggunakan metode analisis deskriptif yang dilakukan dengan proses pengamatan langsung dan mempelajari dari hasil observasi atas hasil yang terjadi dari analisis sistem ini. Data Primer yang digunakan yaitu waktu *rendering* yang didapat pada beberapa kondisi infrastruktur dan animasi dalam eksperimen di laboratorium. Animasi yang diujikan pada penelitian ini sebanyak 3 buah yang dibuat dengan *software* Blender dan memiliki spesifikasi seperti terlihat pada tabel 1.

Alat yang digunakan dalam penelitian ini akan menggunakan alat bantu DrQueue dengan melakukan *distributed rendering* di dalam sistem *cluster* yang rancangan arsitekturnya dapat dilihat pada Gambar 2. Sedangkan spesifikasi perangkat keras dan sistem operasi yang digunakan yaitu sebagai berikut: 1) *5 buah Personal Computer (PC) dengan spesifikasi:* Intel Core 2 Duo; Memory 2 GB DDR2 RAM; Hardisk 60 GB; LAN Card; 2) *Sistem operasi:* Linux Ubuntu v.10.04 – Lucid Lynx; 3) *Switch* 100Mbps (24 Port) ; 4) *Software yang digunakan:* SSH (Secured Shell), NFS (Network File System), DrQueue, Blender, tcsh, scons, libgtk2.0-dev, g++, python, gcc

Tabel 1
Spesifikasi Animasi yang diujikan

Spesifikasi	Animasi		
	Animasi Kubus Sederhana	Animasi MTI	Animasi Ruang Keluarga
Jumlah Frame	100	300	100
Kecepatan Animasi	25	25	25
Resolusi Setiap Frame	800 x 600	800 x 600	800 x 600
Jumlah Vertex	8	40032	885809
Cahaya	1	4	6
Kamera	1	1	1
Kompresi Citra	JPEG	JPEG	JPEG



Gambar 2. Rancangan Arsitektur Sistem *distributed rendering*

Penelitian yang dilakukan melalui tahapan-tahapan seperti dijelaskan sebagai berikut: a) Identifikasi permasalahan yang terjadi pada sistem *rendering* yang hanya menggunakan satu buah komputer; b)Membangun sebuah sistem *distributed rendering* yang terdiri dari: Instalasi sistem operasi Ubuntu 10.04 dalam kondisi *fresh install*; Instalasi dan konfigurasi NFS dan SSH sebagai penunjang komunikasi Komputer *master node* dan *slave node*; Instalasi dan konfigurasi Dr. Queue; Instalasi dan konfigurasi Blender; c) melakukan *rendering* 3 buah *file* film animasi menggunakan DrQueue dan Blender pada 5 kondisi, yaitu 1 Komputer master *node*; 1 Komputer master node dan 1 *slave node*; 1 Komputer master *node* dan 2 *slave node*; 1 Komputer master *node* dan 3 *slave node*; 1 Komputer master *node* dan 4 *slave node;* d) pengujian dengan menggunakan 1 PC Quadcore stand alone; e) dokumentasi dari setiap proses inisialisasi terhadap objek penelitian yang dilakukan dalam menyelesaikan setiap permasalahan; dan f) analisis hasil penelitian.

4. HASIL PENELITIAN DAN PEMBAHASAN

Sistem *Cluster Distributed Rendering* ini dibuat menggunakan lima buah komputer dengan salah satu komputer digunakan sebagai *master node* dan empat komputer lainnya sebagai *slave node*. Masing-masing *node* menggunakan sistem operasi Linux Ubuntu 10.04 (Lucid Lynx) dalam kondisi *fresh install* sehingga dapat dipastikan bahwa dalam sistem operasi yang digunakan tidak terdapat aplikasi-aplikasi yang tidak dibutuhkan dalam sistem *distributed rendering* ini.

4.1 Implementasi Sistem cluster distributed rendering

Pada dasarnya untuk dapat menjalankan sistem *cluster distributed rendering*, semua node yang terhubung pada sistem cluster harus dapat *berkomunikasi* dengan baik. Oleh karena itu, yang harus dilakukan adalah memastikan konektivitas antar *node* sangat baik.

Seperti telah dijelaskan sebelumnya bahwa sistem operasi yang digunakan adalah Ubuntu Lucid Lynx dalam keadaan *fresh install*. Selain kondisi sistem operasi

yang *fresh install* semua *service* yang tidak dibutuhkan sebaiknya dibuang. Selanjutnya, tahapan yang harus dilalui seperti dijelaskan berikut ini:

1) Identifikasi Hostname

Agar setiap *node* dapat saling mengenali, maka setiap node harus diberikan identitas berupa *IP Address* dan *Hostname*. *Hostname* didefinisikan pada *master node* dengan melakukan konfigurasi pada /etc/hosts, sedangkan pada *slave node hostname* yang didefinisikan cukup *hostname* master *node* dan *hostname slave node* tersebut.

Konfigurasi IP Address dilakukan dengan melakukan konfigurasi pada /etc/network/interfaces. Konfigurasi ini dilakukan pada semua *node*, baik master *node* maupun *slave node* dengan menyesuaikan pada *IP Address* masing-masing *node*.

2) Konfigurasi Shared Directory NFS (Network File System)

NFS merupakan sistem *shared directory* yang memungkinkan setiap *node* yang terhubung pada jaringan tersebut membagi (*share*) *directory* pada komputer yang telah ditentukan. Semua *node* di dalam jaringan yang mengaplikasikan NFS dapat mengakses *shared directory* pada NFS *server* dengan cara melakukan *mounting* NFS tersebut pada NFS *client*. NFS yang telah dilakukan proses *mounting*, dianggap oleh NFS *client* sebagai *local file system.*

Pada sistem *cluster distributed rendering*, NFS berfungsi untuk melakukan *shared directory* yang akan ditempatkan *middleware* DrQueue dan *storage directory* yang merupakan tempat *file* hasil *rendering*. Oleh karena itu, dengan adanya NFS, instalasi DrQueue nantinya cukup dilakukan hanya pada master *node*.

Konfigurasi NFS yang dilakukan pada master *node* ada pada tiga lokasi *file* konfigurasi yaitu: /etc/exports yang akan melakukan identifikasi *shared directory* yang diizinkan untuk diakses; /etc/hosts.allow yang akan melakukan identifikasi hosts yang diizinkan untuk melakukan *sharing*; dan /etc/hosts.deny untuk mengidentifikasi *hosts* yang tidak diizinkan untuk melakukan *sharing directory*. Pada *slave node* file konfigurasi NFS terdapat pada *file* /etc/fstab, yang didalamnya dilakukan konfigurasi untuk mendefinisikan *directory* lokal yang digunakan untuk melakukan *mounting* dari *directory server*.

Directory yang perlu dilakukan *sharing* pada sistem *distributed rendering* ini yaitu *directory* yang digunakan untuk meletakkan hasil *rendering* dan *directory* instalasi *middleware* DrQueue.

```
#!/bin/bash
Export DRQUEUE_ROOT = /usr/share/drqueue
Export DRQUEUE_TMP = /usr/share/drqueue/tmp
Export DRQUEUE_MASTER = 172.24.14.184
/usr/share/drqueue/bin/slave 1> /dev/null 2> & 1 &
```

Gambar 1: *Script sinkronisasi pada slave node*

4.2 Pengujian dan Analisis

Tujuan perancangan pengujian *cluster distributed rendering* ini untuk menguji kemampuan dalam melakukan proses komputasi. Parameter yang digunakan adalah s*peedup*, efisiensi dan waktu *rendering*. Pengujian *cluster* menggunakan *tool* DrQman

yang juga merupakan GUI untuk melakukan penugasan pada *middleware* DrQueue.

Pengujian pada setiap animasi dilakukan dengan melakukan *rendering* dari awal hingga akhir *frame* dan sesuai dengan resolusi yang telah diberikan. *Rendering* yang dilakukan pertama dengan satu buah *node* yang kemudian ditambah satu buah *node* pada setiap pengujian.

Dari pengujian terhadap ketiga animasi tersebut didapatkan waktu *rendering* yang dihasilkan dari sistem *distributed rendering* ini pada ketiga animasi yang diujikan. Setelah didapatkan hasil berupa waktu *rendering* setiap kenaikan jumlah *node* waktu tadi kemudian dihitung besar *speedup*-nya dengan cara membagi waktu eksekusi sekuensial dengan waktu eksekusi paralel yang didapat dari waktu *rendering*. Dari hasil perhitungan *speedup* tersebut kemudian dihitung besarnya efisiensi komputasi pada sistem *distributed rendering* ini dengan cara membagi nilai *speedup* yang didapat dengan banyaknya node, seperti telah dibahas pada Persamaan (2). Hasil dari pengujian ketiga parameter ini dapat dilihat pada Tabel II.

Tabel 2
Data Hasil Pengujian Sistem Distributed Rendering

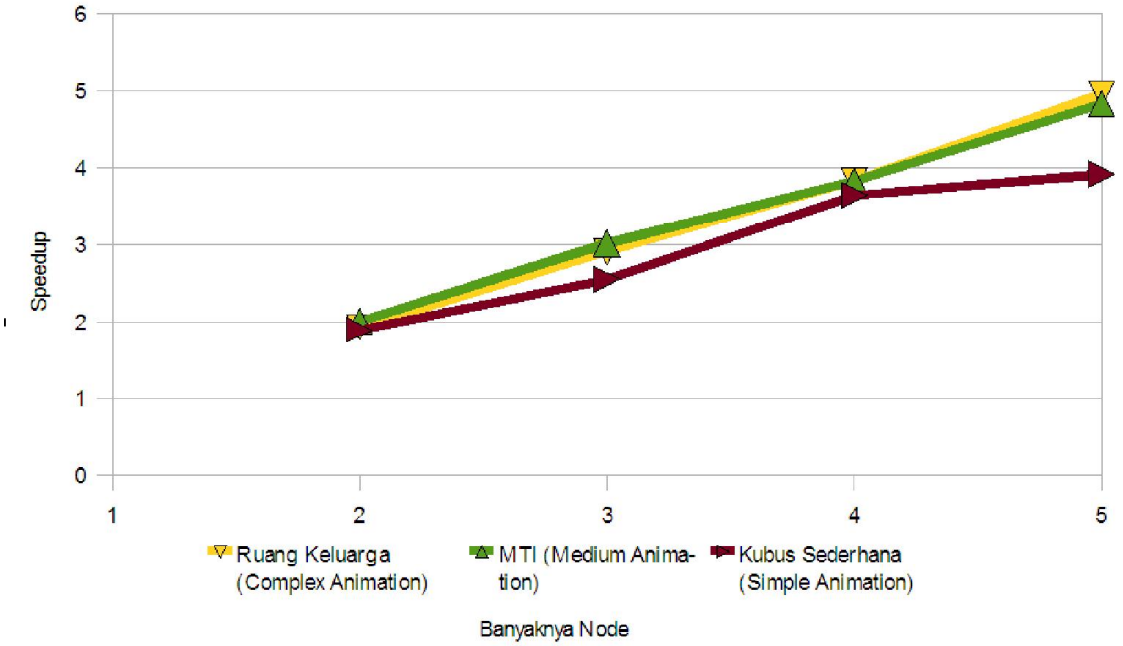
Parameter	Pengujian	Animasi		
		Kubus Sederhana	MTI	Ruang Keluarga
Waktu Rendering (detik)	1 *node distributed rendering*	51	1522	87975
	2 *node distributed rendering*	27	758	45632
	3 *node distributed rendering*	20	505	30182
	4 *node distributed rendering*	14	397	22946
	5 *node distributed rendering*	13	315	17720
Speedup	2 *node distributed rendering*	1,899	2,008	1,928
	3 *node distributed rendering*	2,550	3,014	2,915
	4 *node distributed rendering*	3,643	3,834	3,834
	5 *node distributed rendering*	3,923	4,832	4,970
Efisiensi (x 100%)	2 *node distributed rendering*	0,944	1,004	0,964
	3 *node distributed rendering*	0,850	1,005	0,972
	4 *node distributed rendering*	0,911	0,958	0,959
	5 *node distributed rendering*	0,785	0,966	0,990

Dari Tabel 2 dibuat grafik untuk melakukan analisis sistem *distributed rendering* ini seperti terlihat pada Gambar 4 dan Gambar 5, yang kemudian dapat dilakukan analisis sebagai berikut:

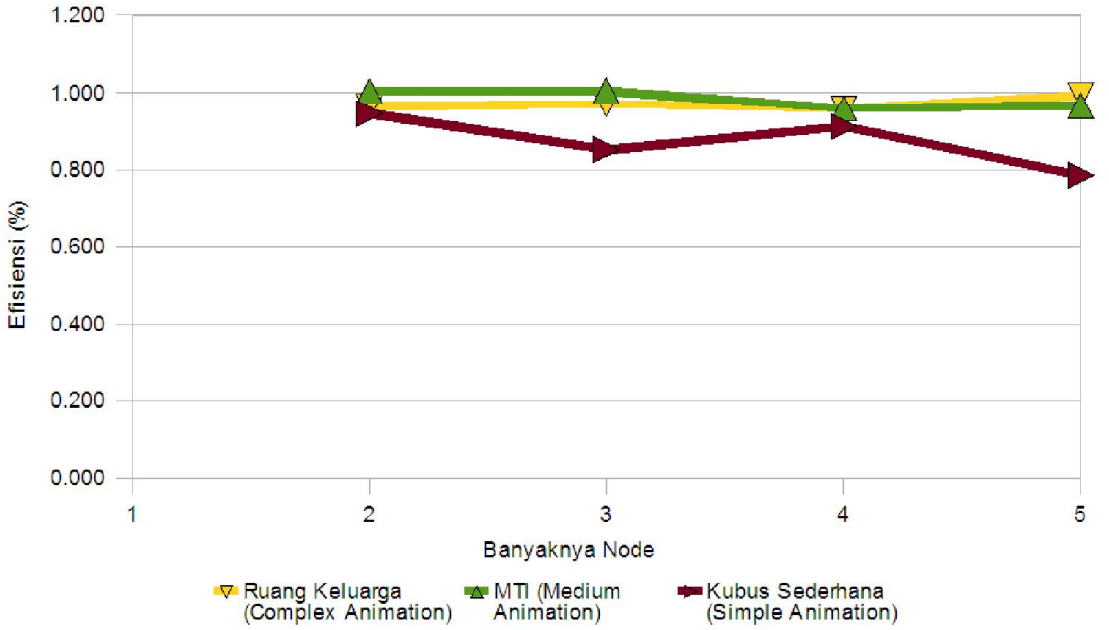
1. Pada animasi Kubus Sederhana, penambahan node sebanyak 2 node memberikan nilai speedup sebesar 1,899; ketika jumlah node ditambah menjadi 3 maka nilai speedup meningkat menjadi 2,550; begitu pula ketika dilakukan penambahan menjadi 4 node, nilai speedup yang dihasilkan adalah sebesar 3,643; dan ketika ditambahkan menjad 5 node nilai speedup yang dihasilkan sebesar 3,923. Dari percobaan tersebut terlihat bahwa dengan penambahan

jumlah node, maka nilai speedup juga akan meningkat.

2. Pada animasi MTI, penambahan node sebanyak 2 node memberikan nilai speedup sebesar 2,008; ketika jumlah node ditambah menjadi 3 maka nilai speedup meningkat menjadi 3,014; begitu pula ketika dilakukan penambahan menjadi 4 node, nilai speedup yang dihasilkan adalah sebesar 3,834; dan ketika ditambahkan menjadi 5 node nilai speedup yang dihasilkan sebesar 4,832. Dari percobaan tersebut terlihat bahwa dengan penambahan jumlah node, maka nilai speedup juga akan meningkat.
3. Pada animasi Ruang Keluarga, penambahan node sebanyak 2 node memberikan nilai speedup sebesar 1,928; ketika dilakukan penambahan menjadi 4 node, nilai speedup yang dihasilkan adalah sebesar 3,834; dan ketika ditambahkan menjadi 5 node nilai speedup yang dihasilkan sebesar 4,970. Dari percobaan tersebut terlihat bahwa dengan penambahan jumlah node, maka nilai speedup juga akan meningkat.
4. Kenaikan kecepatan terjadi cukup signifikan ketika terjadi penambahan dari 1 node menjadi 2 node. Namun ketika node bertambah dari 2 node menjadi 3 node, 4 node, hingga 5 node pertambahan kecepatan yang terjadi tidak terlalu signifikan. Hal ini dikarenakan dari setiap node yang bekerja tersebut terdapat komunikasi data dari slave node ke master node yang memenuhi traffic jaringan.
5. Dari grafik yang terlihat pada Gambar 4.18, ketika penambahan node dari 4 node ke 5 node terjadi kenaikan speedup pada animasi Ruang Keluarga, hal ini tentunya berpengaruh kepada persentase efisiensi seperti terlihat pada Gambar 4.19. Sehingga dapat dikatakan bahwa semakin kompleks animasi dan node yang bekerja persentase efisiensi akan bertambah.



Gambar 4. Grafik nilai Speedup Pada Keempat Animasi yang Diujikan



Gambar 5. Grafik nilai Efisiensi Pada Keempat Animasi yang Diujikan

1. KESIMPULAN

Dari hasil penelitian ini dapat ditarik beberapa kesimpulan, diantaranya: 1) dalam merancang sistem *distributed rendering* ini diperlukan beberapa PC yang saling terhubung dalam suatu jaringan yang didedikasikan untuk sistem *distributed rendering* ini. Hal ini dikarenakan sistem *distributed rendering* yang menggunakan *middleware* DrQueue ini membutuhkan *permission* yang memungkinkan PC lain melakukan *full access* pada *directory* tertentu pada suatu PC, untuk kepentingan *storage directory* hasil *rendering*. Selain itu sistem akan lebih optimal ketika pada *slave node* dan master *node*, *middleware* DrQueue diatur agar menjadi aplikasi *startup*. Hal ini untuk mengantisipasi agar DrQueue dapat melanjutkan tugasnya ketika *node* yang bersangkutan mengalami *restart* secara tiba-tiba; 2) Pada ketiga animasi yang diujikan, kenaikan *speedup* terjadi cukup signifikan ketika terjadi penambahan dari 1 *node* menjadi 2 *node* yaitu rata-rata sebesar 89%. Namun, ketika *node* bertambah dari 4 *node*, hingga 5 *node* pertambahan speedup yang terjadi tidak terlalu signifikan yaitu rata-rata sebesar 21%. Hal ini dikarenakan dari setiap *node* yang bekerja tersebut terdapat komunikasi data dari *slave node* ke *master node* yang memenuhi *traffic* jaringan sehingga berakibat pada terjadinya titik jenuh pada kenaikan *speedup*

DAFTAR PUSTAKA

Eager, D. L; Zoharjan, J; & Lazowska, E. D. 1989. Speedup Versus Efficiency in Parallel System. *IEEE Transaction On Computer*, Vol. 38, No. 3, March 1989.

Grama, A; Gupta, A.L; Karypis, G; & Kumar, V. 2003. *"Introduction to Parallel Computing"*. London: Addison Wesley.

Jamal, A & Sistha, P. 2006. Kinerja Komunikasi Data Kolektif Broadcast pada PC cluster, *Risalah Lokakarya Komputasi dalam Sains dan Teknologi Nuklir XVII*, Jakarta, Indonesia

Laksono, A. D.; Mutiara, A. B.; Heruseto, Brahmantyo. 2004. Analisis perbandingan antara cluster openmosix dengan MPI terhadap aplikasi rendering POV-RAY*, Proceedings Komputer dan Sistem Intelijen*. Depok: Universitas Gunadarma

Putri, D. F. M.; Jatmiko, S; Mutiara, A. B. 2004. Perbandingan Kinerja cluster OpenMosix dengan Disk dan Tanpa Disk*, Proceedings Komputer dan Sistem Intelijen*. Depok: Universitas Gunadarma