Studi Skalabilitas Pemrosesan Paralel Pada Sistem Terdistribusi

Uray Yufikar
Fakultas Teknik, Program Studi Informatika
Universitas Langlangbuana
Jl. Karapitan No. 116, Bandung
Email : yufikar@gmail.com

Abstrak **-** Salah satu keistemewaan aplikasi yang menggunakan pemrosesan paralel adalah skalabilitas, yaitu bisa dipercepat, atau bila beban komputasinya meningkat maka performansinya dapat dipertahankan dengan menambah pemrosesnya. Pemrosesan paralel idealnya dijalankan pada komputer paralel, atau *multi processor*, namun komputer seperti ini masih belum menjadi komoditas. Hadirnya komputer dengan prosesor *multicore* adalah solusi, sehingga pemrosesan paralel dalam aplikasi menjadi lazim saat ini, walau skalabilitasnya tentu sangat terbatas, karena kita tidak dapat menambah *processor* atau *core* pada komputer tersebut. Studi ini memperlihatkan bahwa melakukan proses paralel pada sekumpulan komputer yang terkoneksi dalam sistem terdistribusi adalah suatu alternatif, karena kita dapat menambah komputer disana untuk mendapatkan skalabilitas.

Kata kunci **-** *multi-processor*, *processor*, *core. multi-core*

1. PENDAHULUAN

1.1. Latar Belakang

Pemrosesan paralel membutuhkan komputer yang mempunyai beberapa prosesor atau biasa disebut dengan komputer paralel [1]. Ada 2 basis arsitektur komputer paralel, yang pertama adalah berbasis *Memory Sharing* dimana setiap prosesornya mengakses *memory address space* yang sama, yang kedua adalah berbasis *Distributed Memory* [2] dimana setiap prosesornya mempunyai *memory* lokal masing-masing. Komputer dengan prosesor *multicore* yang umum kita pakai termasuk golongan *Memory Sharing*. Komputer paralel dengan arsitektur *Distributed Memory* yang komersial diantaranya adalah UNIX CLUSTER SYSTEM, Komputer ini biasanya dipakai untuk keperluan HPC (High Performance Computing) [3] misal server berformansi tinggi atau untuk keperluan *scientific*. Komputer dulu yang prosesornya *single-core* hanya bisa melakukan proses paralel secara *pseudo*, atau dikenal dengan nama pemrosesan konkuren atau *multi-thread.[3]*

Di luar kedua jenis pemrosesan pada komputer diatas, sistem pelayanan komputer secara jaringan (*computer network services*) terus berkembang, Saat ini sistem pelayanan jaringan komputer secara terdistribusi atau biasa disebut *Distributed System*, telah menggantikan sistem pelayanan jaringan komputer secara terpusat atau *Centralized System*. Pada Sistem-Terdistribusi pelayanan bisa dilakukan oleh satu atau sekumpulan komputer *independent* yang bekerjasama. Adanya mekanisme kerjasama antar komputer ini memberi peluang menggunakan sekumpulan komputer yang terkoneksi dengan Sistem-Terdistribusi melakukan proses paralel mencontoh UNIX CLUSTER SYSTEM. Hal itulah yang menjadi latar belakang studi ini. [2]

1.2. Maksud

Pada Sistem-Terdistribusi komputer-komputer bisa terpisah secara *geography* maupun perbedaan *platform [1].* Maka pada mekanisme kerjasama antar komputernya pasti melalui tahapan konversi antar representasi data untuk mengatasi *heterogenitas*, kemudian tahapan untuk mengakomodasi *communication delay and fail [2]*. Dengan demikian lingkungan Sistem-Terdistribusi adalah *asynchronous* karena tidak memungkinkan adanya kerangka waktu bersama, tentu ini bukan lingkungan ideal untuk pemrosesan paralel yang membutuhkan kerangka waktu *synchronous*. Studi ini dimaksudkan untuk mengamati apakah batasan tersebut dan kendala lain yang dijumpai baik saat pemrograman maupun implementasi tidak mengurangi kelayakan melakukan proses paralel pada Sistem-Terdistribusi.[2]

1.3. Tujuan

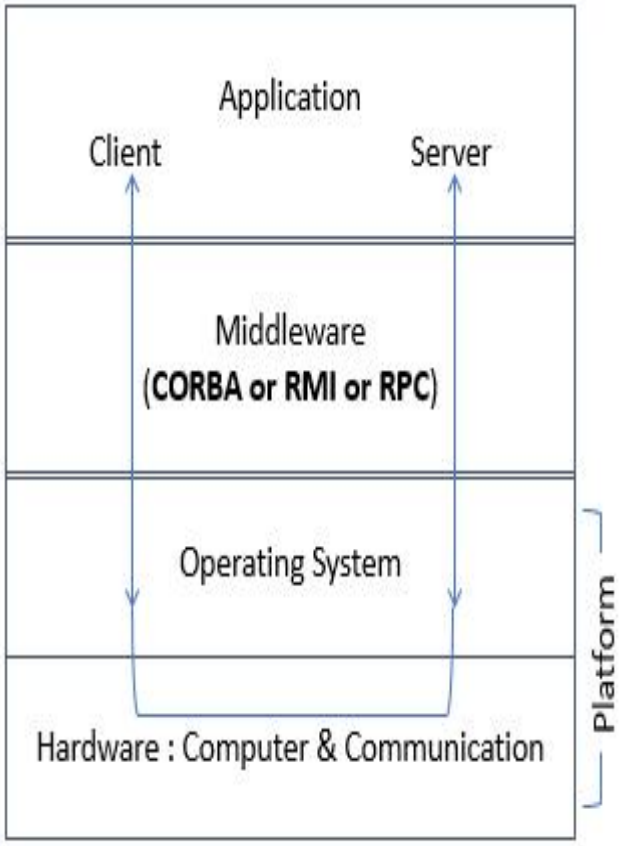
Kecepatan pemrosesan paralel selain diperoleh dari efisiensi algoritmanya, juga dari jumlah pemrosesnya, Studi ini tujuannya bukan pengamatan terhadap efisiensi algoritma, melainkan pada aspek skalabilitasnya, yaitu kemudahan menambah pemroses bila diperlukan. Aspek yang mendukung skalabilitas pemrosesan paralel pada Sistem-Terdistribusi yang akan diamati misalnya, apakah penambahan komputer untuk mempercepat proses juga membutuhkan modifikasi *source-code* dan bagaimana membagi beban pemrosesan apakah bisa dilakukan secara dinamis. Contoh kasus yang dipilih pada studi ini adalah pengurutan data dengan metoda *merge-sort*, memang bukan algoritma terbaik dalam *data-sorting*, melainkan lebih dapat mewakili kasus pemrosesan paralel dan memudahkan visualisasi aspek skalabilitasnya.

2. ULASAN

2.1. Middleware Sistem-Terdistribusi

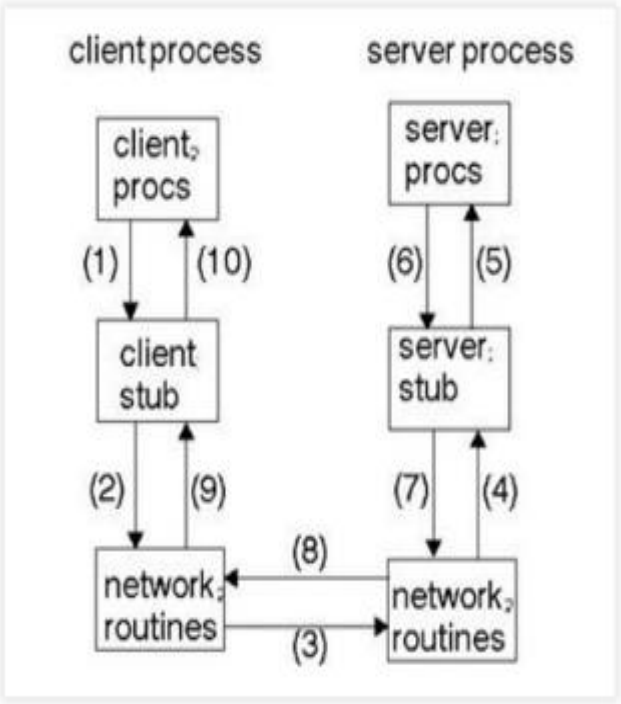
Secara *software,* arsitektur Sistem-Terdistribusi terdiri 4 lapisan seperti terlihat pada Gambar 1, lapisan kedua, *Middleware,* adalah pemegang peranan utama dalam membentuk Sistem-Terdistribusi [2]. *Middleware* adalah jembatan yang menguhubungkan antara aplikasi *client* dengan aplikasi *server* di jaringan, kerangka kerjanya adalah sbb :

a. Melakukan konversi representasi data yang dianut di *client* menjadi format data standard dari *Sistem Message* yang dianut.
b. Memaket data tsb. kedalam *buffer* dari *network protocol* yang dianut.
c. Mengirim paket, sebagai *message request* dari *client* ke *server.*
d. Menunggu penyelesaian proses, kemudian memaket dan mengirim kembali hasilnya sebagai *message respon* dari *server* ke *client*.
e. Melakukan konversi balik *message respon* ke representasi data yang dianut di client



Gambar 1. Arsitektur Softwere Sistem-Terdistribusi

Teknologi *middleware* masih terus berkembang, namun mengarah ke 3 aliran besar yaitu : RPC (Remote Procedure Call), CORBA (Common Object Request Broker Architecture) dan RMI (Remote Method Invocation). Yang dipakai dalam studi ini adalah RPC. Representasi RPC di dalam aplikasi adalah berupa *client stub* dan *server stub* seperti terlihat pada Gambar 2, cara kerja detilnya terdiri 10 langkah yang dapat dibaca pada berbagai referensi lain, namun secara global akan sesuai dengan kerangka kerja *middleware* umumnya seperti tertulis diatas.[2]



Gambar 2. Cara Kerja RPC

2.2. Memberikan Kemampuan RPC Pada Aplikasi

Banyak paket RPC siap pakai, diantaranya : Microsoft-RPC, xmlRPC, googleRPC, libtirpc, RPCLIB dll. Semua paket tersebut adalah berupa *library* yang akan dihubungkan atau lebih tepatnya digabung (*embedding*) melalui proses *compilation & link* dengan aplikasi yang kita buat. Setelah proses *embedding* maka aplikasi kita akan mempunyai fungsi-fungsi untuk melaksanakan komunikasi dan pertukaran data atau kemampuan melakukan mekanisasi *middleware*, disisi aplikasi *client* kemampuan itu disebut *client-stub*, dan disisi aplikasi *server* disebut *server-stub.* Jadi kedua *stub* itu dalam kenyataannya bukan terlihat sebagai aplikasi sendiri sebagaimana Gambar 2, melainkan kemampuan yang menyatu dalam aplikasi *server* atau *client* yg kita buat.[2]

2.3. Menggunakan Paket RPCLIB

Paket *library* -RPC yang dipakai di studi ini adalah RPCLIB. Paket ini dipilih karena kemudahannya, Dengan RPCLIB kita hanya butuh 2 langkah untuk memberikan kemampuan *stub* pada aplikasi *server* dan aplikasi *client [2],* yaitu sbb :

a. Sertakan *header-stub* pada *source-code* aplikasi *server* juga aplikasi *client*
b. Pada aplikasi *server*, binding semua prosedur yang akan di *share* ke *client*.
c. Pada aplikasi *client,* gunakan perintah *call* untuk mengakses/mengeksekusi prosedur yang ada di server.

Contoh Aplikasi singkat untuk *client* dan *server* dapat dilihat pada gbr. 3 dibawah ini. Client memanggil/mengeksekusi prosedur bernama SrvGetTxt dengan memberikan input parameter yang akan tampil di layar server, kemudian server membalas teks yang akan ditampilkan dilayar *client.*

```
#include <iostream>
#include "rpc/client.h"

int main()
{
  // crt & connect cln-stub to svr-stub at IP,Port
  rpc::client clientStub("127.0.0.1", 8081);

  // call srv 'procedure' with 'parameter' and get the return as 'result'
  auto result = client.call("SrvGetTxt", "Hello Its From Client").as<std::string>();

 // display the return text from server
 std::cout << "The result is: " << result << std::endl;

  return 0;
}
```

```
#include <iostream>
#include "rpc/server.h"

std::string SrvGetTxt(std::string capt) {
  std::cout << capt ;
  return("Hello.. Its from Server");
}

int main()
{
  // Create Server-Stub at IP,Port
  rpc::server srv("0.0.0.0", 8081);

  // binding the procedure
  srv.bind("SrvGetTxt", &SrvGetTxt);

  // start
  srv.run();

  return 0;
}
```

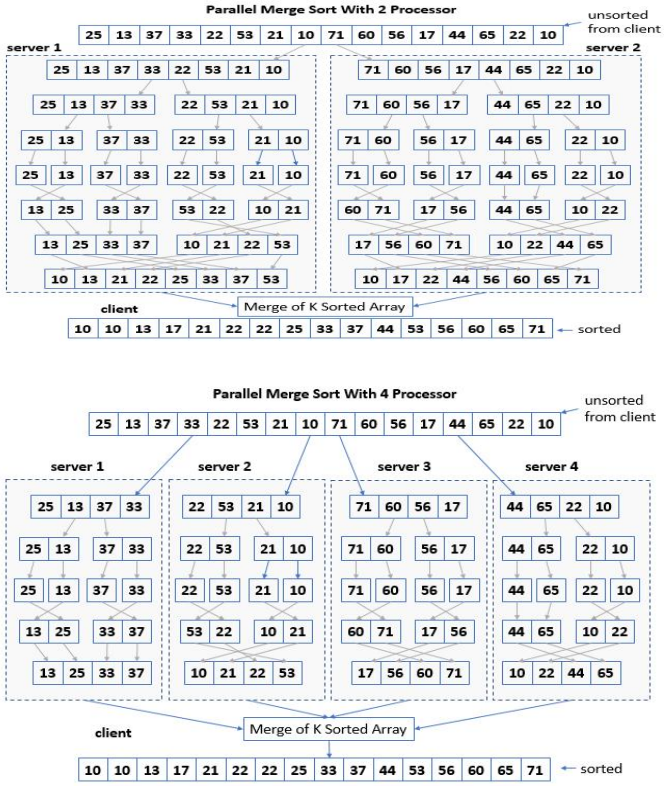
Gambar 3. Contoh Singkat Aplikasi *Client & Server [2]*

Diatas dapat dilihat bahwa membuat aplikasi Sistem-Terdistribusi dengan paket RPCLIB, sangat transparan dengan pemrograman C++, dalam membuat aplikasi *stand-alone* biasa, hanya menambahkan perintah-perintah terkait *stub* saja. Tidak banyak yang harus dipelajari, sebagai perbanding, untuk membangun aplikasi Sistem-Terdistribusi dengan paket MicrosoftRPC, kita perlu belajar IDL (*Interface Definition Language*) untuk mendefenisikan interface antar *client* dan *server*-nya.[2]

3. DESAIN PROSES

3.1. Pemrosesan Paralel Merge-Sort

Yang akan dijadikan contoh pemrosesan paralel pada Sistem-Terdistribusi di studi ini adalah pengurutan data dengan metode *merge-sort*, Pada metoda tersebut data yang akan diurutkan disimpan dalam *vector* (*array* dinamis), kemudian *vector* tersebut di bagi dua terus menerus hingga masing-masing *vector* hanya mempunyai satu elemen, langkah selanjutnya setiap dua *vector* berdampingan digabung kembali tetapi secara terurut, ini dilakukan terus menerus hingga menjadi satu *vector* yang elemennya sudah terurut. Dalam studi ini proses memecah dan menggabung kembali *vector-vector* tsb. akan dibagikan kepada sejumlah server, sehingga masing-masing bisa melakukannya bersamaan. Gambar 4 dibawah ini memperlihatkan proses *merge-sort* dengan 2 dan 4 server.

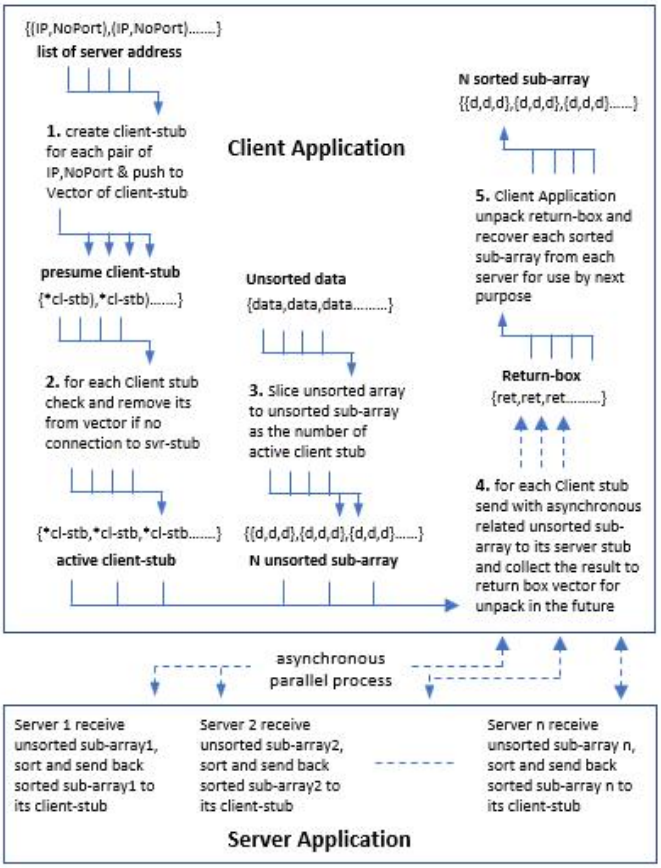


Gambar 4. Pemrosesan Paralel Merge-Sort dengan 2 & 4 Server [2]

3.2. Skenario Proses Pada Aplikasi Server

Tidak banyak tahapan proses di *main program* aplikasi *server,* hanya terdiri atas beberapa langkah protokoler *stub* saja. Proses utama di aplikasi *server* ada didalam prosedur yang dipanggil/dieksekusi oleh aplikasi *client*, Pada studi ini proses tersebut adalah mengurutkan data dengan metoda *merge-sort,* yang skenario/algoritma prosesnya tidak dijelaskan lagi disini karena sudah umum dan standard [2]. Skenario proses aplikasi server adalah sbb :

a. Membuat *instance server-stub* dengan IP dan *port-number* yang diberikan
b. Mem-*binding* prosedur yang akan di-*share* kepada *client* pada *server-stub*
c. Menjalankan *server-stub* untuk mulai menunggu (*listen*) perintah dari *client*
d. Menjalankan prosedur merge-sort dengan data yang diterima dari *client*
e. Mengembalikan data hasil *sort* kepada *client* dipenampung yg disediakan



Gambar 5. Skenario proses di aplikasi *client [2]*

3.3. Skenario Proses Pada Aplikasi Client

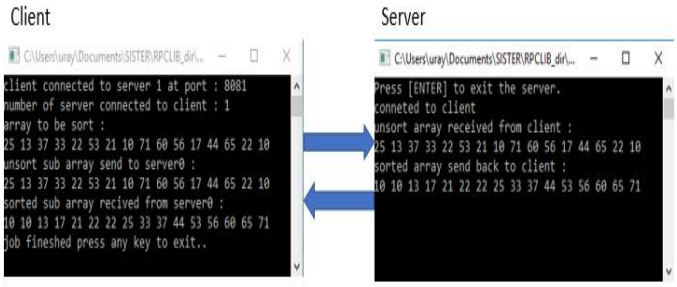
Skenario proses pada aplikasi *client* adalah seperti pada gambar 5 diatas, yaitu :

a. Membaca daftar alamat IP & PortNo dari *server*.
b. Membuat sejumlah *instance client-stub* untuk masing-masing IP & PortNo tersebut dan menyimpannya pada *vector client-stub* (1).
c. Melakukan check koneksi masing-masing *client-stub,* yang tidak terkoneksi pada pasangannya *server-stub* dimusnahkan dari *vector client-stub* (2)
d. Baca data yang akan diproses dan menyimpannya ke *vector unsorted array*
e. Bagi *vector unsorted array* menjadi *vector unsorted sub-array,* sebanyak *client-stub* yang aktif (3)
f. Setiap client-stub melakukan pengiriman data *vector unsorted sub-array* bagiannya masing-masing kepada *server-stub* pasangannya, untuk di *sort* disana, sambil memberitahukan tempat penampungan paket balik hasilnya (4).
g. Membuka paket-paket yang ada di penampungan menjadi *vector sorted sub-array* yang jumlahnya pasti sesuai dengan jumlah *client-stub* (5).

4. SCREEN CAPTURE HASIL *RUNNING*

4.1. Pemrosesan dengan satu server

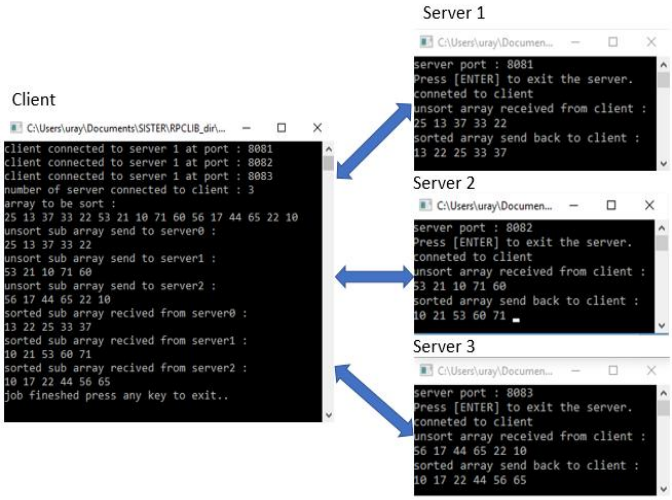
Karena *server* hanya satu, maka aplikasi di *client* mengirimkan *unsorted array* lengkap tanpa dibagi-bagi hanya ke *server* tsb, dan menerima kembali hasil- nya berupa sebuah *sorted array* secara lengkap juga seperti gambar 6 dibawah.



Gambar 6. Hasil *running* dengan 1 *server*

4.2. Pemrosesan dengan 3 server

Karena ada 3 *server* yang terkoneksi ke *client*, maka aplikasi di *client* membagi *Unsorted Array* menjadi 3 *unsorted sub-array*, dan masing-masing dikirim ke *server* berbeda, dan hasilnya yang diterima kembali oleh client adalah *3 sorted sub-array* (siap digabung menjadi total sorted array), seperti dibawah ini



Gambar 7. Hasil *running* dengan 3 *server*

5. ANALISIS HASIL STUDI

Dua contoh hasil *running* diatas memperlihatkan bahwa, pemrosesan paralel bisa dilakukan pada Sistem-Terdistribusi. Hasil tersebut juga memperlihatkan bahwa beban komputasi bisa dibagikan kepada sejumlah komputer secara dinamis, tanpa perlu modifikasi program, dengan demikian skalabilitas pemrosesan paralel pada Sistem-Terdistribusi adalah sangat tinggi.

Dari studi ini didapat beberapa kondisi yang perlu menjadi perhatian bila akan melaksanakan pemrosesan paralel pada Sistem-Terdistribusi, yaitu masalah pertukaran data (*data interchange*) antar komputer. Kita tahu bahwa pertukaran data pada Sistem-Terdistribusi khususnya yang menggunakan RPC adalah melalui *call-procedure* dimana data yang dikirim oleh *client* ke *server* adalah merupakan *parameter-function* dari suatu fungsi, kemudian data balik yang diterima oleh *client* dari *server* adalah *return function* dari fungsi tersebut, seperti contoh dibawah ini :

```
results.push_back(client-stub -> async_call("mrgsort", sub[clientIndex]));
```

packet data balik hasil proses server yang dikirim ke client

nama prosedur di server yang akan memproses data

data dari client yang akan diproses di server

Parameter pada fungsi diatas harus diberikan dalam bentuk *passing by value*, tidak bisa dengan *passing by reference*, karena prinsip kerja Sistem-Terdistribusi bukan dengan memory sharing. Hal ini menyebabkan ada keterbatasan volume dalam pertukaran data. Dilemanya bila pertukaran data diatur kecil-kecil maka juga tidak efisien dalam frekwensi transfernya karena pada Sistem-Terdistribusi itu dilakukan melalui jaringan bukan lewat bus. Maka kompromi pemrosesan paralel yang bagus dilaksanakan pada Sistem-Terdistribusi adalah untuk aplikasi yang sifatnya bukan *interactive-process*, yang banyak kordinasi antar proses, juga bukan pemrosesan yang perlu pengiriman dan penerimaan data sekaligus dalam jumlah besar (*Batch*).

6. PENUTUP

6.1. Kesimpulan

Pemrosesan paralel bisa dan layak dilakukan pada sekumpulan komputer di jaringan Sistem-Terdistribusi, terutama untuk aplikasi yang padat proses tetapi datanya relatif sedikit misal bidang *scientific, cryptology,* atau untuk proses yang ujungnya terbuka tidak ada proses lanjut misalnya *search engine*.

Penambahan komputer untuk mempercepat proses atau untuk mempertahankan performansi pemrosesan paralel pada Sistem-Terdistribusi mudah dilakukan, bahkan bisa dilakukan secara dinamis saat aplikasi sudah berjalan, jadi bisa disebut skalabilitasnya tinggi.

6.2. Saran

Keutamaan Sistem-Terdistribusi adalah heterogenitas atau kemampuan kerjasama antar komputer berbeda *platform*, maka sangat baik bila ada yang melanjutkan studi ini untuk mencoba pemrosesan paralel dengan sekumpulan komputer yang terkoneksi pada jaringan Sistem-Terdistribusi tetapi berbeda *operating system*.

DAFTAR PUSTAKA

[1] Andrews S. Tanenbaum, Maarten Van Steen “Distributed Systems, Principles and Paradigms”, Prentice Hall

[2] By G. Coulouris, J. Dollimore and T. Kindberg “Distributed Systems Concepts and Design” Published by Addison Wesley

[3] Foster and S. Taylor. “New Concepts in Parallel Programming” Prentice-Hall