UNIVERSITAS INDONESIA

PENGEMBANGAN LANJUT *TABLING* PADA *CONTEXTUAL ABDUCTION* DENGAN *ANSWER SUBSUMPTION*

SKRIPSI

SYUKRI MULLIA ADIL PERKASA
1306381793

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JULI 2017

UNIVERSITAS INDONESIA

PENGEMBANGAN LANJUT *TABLING* PADA *CONTEXTUAL ABDUCTION* DENGAN *ANSWER SUBSUMPTION*

SKRIPSI
Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Ilmu Komputer

SYUKRI MULLIA ADIL PERKASA
1306381793

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JULI 2017

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	:	**Syukri Mullia Adil Perkasa**
NPM	:	**1306381793**
Tanda Tangan	:	
Tanggal	:	**21 Juni 2013**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Syukri Mullia Adil Perkasa
NPM	:	1306381793
Program Studi	:	Ilmu Komputer
Judul Skripsi	:	Pengembangan Lanjut *Tabling* pada *Contextual Abduction* dengan *Answer Subsumption*

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Ilmu Komputer pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing	:	Ari Saptawijaya S.Kom., M.Sc., Ph.D.	(	)
Penguji	:	Penguji 1	(	)
Penguji	:	Penguji 2	(	)

Ditetapkan di	:	Depok
Tanggal	:	5 Juli 2013

KATA PENGANTAR

Alhamdulillahirabbil'alamin, segala puji dan syukur kehadirat Tuhan Yang Maha Esa, Allah Subhana Huwataala, karena hanya dengan hidayah dan rahmat-Nya, penulis dapat menyelesaikan pembuatan skripsi ini.

Allahumma sholli 'alaa sayyidina Muhammad, Sholawat serta salam tak henti-hentinya dipanjatkan kepada Rasulullah SAW, atas peranannya di muka bumi dalam memberikan tuntunan kepada seluruh umat manusia, dan sebagai inspirasi atas seluruh manusia sebagai manusia dengan akhlak terbaik.

Penulisan skripsi ini ditujukan untuk memenuhi salah satu syarat untuk menyelesaikan pendidikan pada Program Sarjana Ilmu Komputer, Universitas Indonesia. Saya sadar bahwa dalam perjalanan menempuh kegiatan penerimaan dan adaptasi, belajar-mengajar, hingga penulisan skripsi ini, penulis tidak sendirian. Penulis ingin berterima kasih kepada pihak-pihak berikut :

Depok, 17 Juni 2013

Syukri Mullia Adil Perkasa

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Syukri Mullia Adil Perkasa
NPM	:	1306381793
Program Studi	:	Ilmu Komputer
Fakultas	:	Ilmu Komputer
Jenis Karya	:	Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul:

Pengembangan Lanjut *Tabling* pada *Contextual Abduction* dengan *Answer Subsumption*

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyatan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 21 Juni 2013

Yang menyatakan

(Syukri Mullia Adil Perkasa)

ABSTRAK

Nama	:	acoba
Program Studi	:	Ilmu Komputer
Judul	:	Pengembangan Lanjut *Tabling* pada *Contextual Abduction* dengan *Answer Subsumption*

Abstrak INA

Kata Kunci:
atu, dua, *tiga*

ABSTRACT

Name	:	Syukri Mullia Adil Perkasa
Program	:	Computer Science
Title	:	Advanced Development of Tabling in Contextual Abduction with Answer Subsumption

Abstract in Eng

Keywords:
one,two,three

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Menurut Buyya terdapat 3 buah contoh untuk membuat enumerate pada latex (Buyya, 1999):

1. Makan
2. Minum

Menurut Mozdzynski (2012), pemodelan yang sama apabila dijalankan dengan komputer *Dual Core* maka akan membutuhkan waktu 1 tahun dengan asumsi memori yang dibutuhkan cukup (Mozdzynski, 2012).

1.2 Perumusan Masalah

Pada bagian ini akan dijelaskan mengenai definisi permasalahan yang dihadapi dan ingin diselesaikan serta asumsi dan batasan yang digunakan dalam menyelesaikannya.

1.3 Tujuan dan Manfaat Penelitian

Dibawah ini adalah contoh itemize :

- Terimplementasinya .
- Menyelesaikan masalah .

1.4 Tahapan Penelitian

@todo
Tuliskan tujuan penelitian.

1.5 Ruang Lingkup Penelitian

1.6 Sistematika Penulisan

Sistematika penulisan laporan adalah sebagai berikut:

- Bab 1 PENDAHULUAN
- Bab 2 LANDASAN TEORI
- Bab 3 IMPLEMENTASI
- Bab 4 EVALUASI DAN ANALISIS
- Bab 5 PENUTUP

@todo
Tambahkan penjelasan singkat mengenai isi masing-masing bab.

BAB 2
LANDASAN TEORI

Bab ini berisi dasar-dasar pemahaman yang diperlukan dalam membuat program logika.

2.1 Pendahuluan

Diberikan himpunan alfabet $\mathcal{A}$ dari bahasa $\mathcal{L}$, akan didefinisikan himpunan berhingga yang saling *disjoint* yaitu himpunan konstanta, himpunan simbol fungsi, dan himpunan simbol predikat. Selain itu, pada alfabet juga mengandung himpunan simbol variabel. Simbol *underscore* (_) dikhususkan untuk menyatakan sebuah *anonymous* variabel. *Term* dari $\mathcal{A}$ didefinisikan secara rekursif sebagai salah satu dari: variabel, konstanta, atau ekspresi dengan bentuk $f(t_1,...,t_n)$ dengan f adalah simbol fungsi dari $\mathcal{A}$ dan t_i adalah term. *Atom* dari $\mathcal{A}$ adalah ekspresi dengan bentuk $p(t_1,...,t_n)$ dengan p adalah simbol predikat dari $\mathcal{A}$ dan t_i adalah term. Selanjutnya, notasi p/n digunakan untuk menyatakan predikat p memiliki arity n. *Literal* adalah sebuah atom a atau negasinya *not a*. Literal *not a* (seperti pada bentuk kedua) disebut *default* literal. Sebuah term (atom maupun literal) disebut *ground* jika term tersebut tidak memiliki variabel. Himpunan seluruh ground term dari $\mathcal{A}$ disebut Herbrand Universe dari $\mathcal{A}$.

2.2 Program Logika

Program logika adalah himpunan berhingga dari *rule* dengan bentuk:

$$H \leftarrow L_1, ..., L_n$$

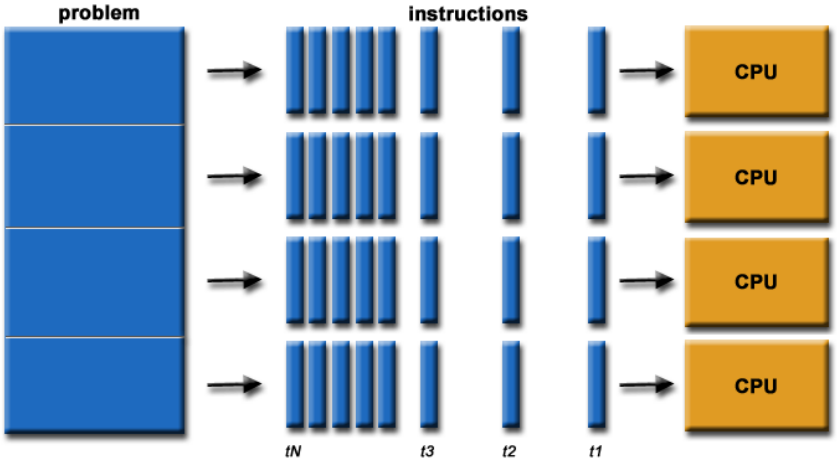
dengan H adalah sebuah *atom*, $m \geq 0$, dan L_i adalah *literal*. H dan L_i berturut-turut disebut sebagai *head* dan *body* dari sebuah rule.

Operator koma pada sebuah rule dibaca sebagai konjungsi. Sebuah program logika disebut *definit* jika pada program logika tersebut tidak mengandung default literal. Rule yang tidak memiliki body ditulis sebagai H saja, alih-alih menuliskannya sebagai $H \leftarrow$. Rule dengan bentuk seperti ini disebut sebagai sebuah *fakta*.

2.2.1 Pengertian X

Setiap gambar dapat diberikan caption dan diberikan label. Label dapat digunakan untuk menunjuk gambar tertentu. Jika posisi gambar berubah, maka nomor gambar juga akan diubah secara otomatis. Begitu juga dengan seluruh referensi yang menunjuk pada gambar tersebut.

Contoh sederhana adalah Gambar 2.1. Silahkan lihat code LaTeX dengan nama bab2-landasan-teori.tex untuk melihat kode lengkapnya. Harap diingat bahwa caption untuk gambar selalu terletak dibawah gambar. Dibawah adda figure, jangn lupa dimention dengan 2.1.



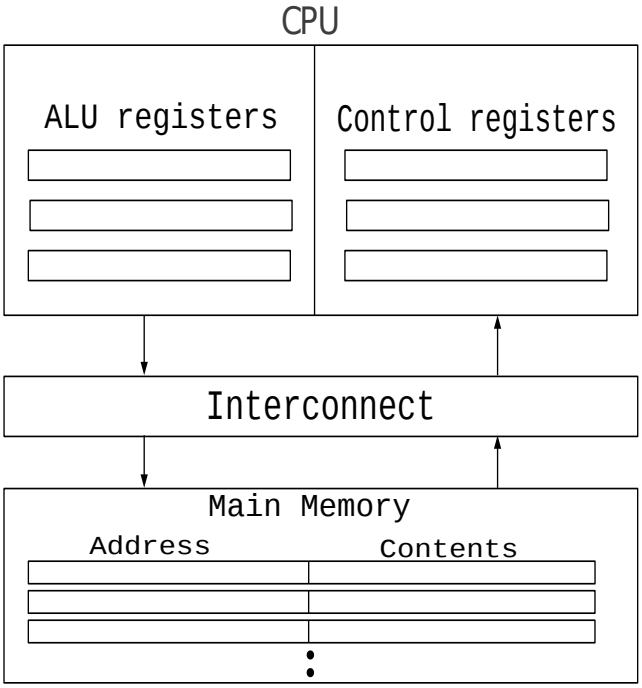
Gambar 2.1: Contoh masalah yang dikerjakan secara paralel

Sumber gambar: (Oxford Dictionaries, 2010)

2.2.2 Klasifikasi X

Figure dalam enum dan dua sitasi sekaligus (Buyya, 1999; Jones et al., 2002) :

1. ***Bold Italic***
   Penjelasan..... Untuk gambarannya dapat dilihat di Gambar 2.2.



Gambar 2.2: Arsitektur klasik *von Neumann*

Sumber gambar terinspirasi dari: (Pressman, 2010)

2. ***Sesuatu banget***
   Penjelasan.....

2.3 *Section in Eng*

Hal pertama yang mungkin ditanyakan adalah bagaimana membuat huruf tercetak tebal, miring, atau memiliki garis bawah. Pada Texmaker, Anda bisa melakukan hal ini seperti halnya saat mengubah dokumen dengan LO Writer. Namun jika tetap masih tertarik dengan cara lain, ini dia:

- **Bold**
  Gunakan perintah \textbf{} atau \bo{}.
- *Italic*
  Gunakan perintah \textit{} atau \f{}.
- Underline
  Gunakan perintah \underline{}.

- $\overline{Overline}$
  Gunakan perintah \overline.
- ^{superscript}
  Gunakan perintah \{}.
- $_{subscript}$
  Gunakan perintah _{}.

Perintah \f dan \bo hanya dapat digunakan jika package uithesis digunakan.

2.3.1 Pengertian *Section in Eng*

2.3.2 Next Subsection *Section in Eng*

2.4 *Keatas lagi*

Contoh cite yang ga ada **?**. Cite author Neal,cite tahun 2004, cite mention Guarddin (2010), dan cite di akhir kalimat (Mell dan Grance, 2009).

2.4.1 *Masuk lagi*

Footnote example nih : MPICH [1], LAM/MPI [2], dan OpenMPI [3] (McGuire, 2010). MPI-3 sedang dalam tahap perencanaan [4]. Fungsi-fungsi tersebut berada di tabel 2.1. (Contoh tabel).

Tabel 2.1: Fungsi fundamental MPI

No.	Nama Fungsi	Penjelasan
1	MPI_Init	Memulai kode MPI
2	MPI_Finalize	Mengakhiri kode MPI
3	MPI_Comm_size	Menentukan jumlah proses
4	MPI_Comm_rank	Menentukan label proses
5	MPI_Send	Mengirim pesan
6	MPI_Recv	Menerima pesan

Sumber tabel: taro sitasi disini, if i were u

[1] http://www.mpich.org/
[2] http://www.lam-mpi.org/
[3] www.open-mpi.org
[4] http://meetings.mpi-forum.org/MPI_3.0_main_page.php

BAB 3
IMPLEMENTASI

Pada bab ini penulis akan menjelaskan bagaimana implementasi *tabling* pada *contextual abduction* dengan *answer subsumption* yang dibuat menggunakan bahasa pemrograman XSB Prolog.

3.1 Alur Program

Pertama akan dijelaskan alur dari program yang dibuat oleh penulis. Secara garis besar, berikut adalah urutan tahap-tahap yang dilakukan oleh program:

1. Melakukan transformasi program input *P* menjadi program output *P'*.
2. Me-*load* program output *P'* ke *environtment* XSB.
3. Melakukan proses *abduction* dengan memberikan query yang dapat disertai dengan *abductive context*.

Penjelasan lebih detil mengenai setiap langkah penulis jabarkan pada subbab-subbab berikutnya.

3.2 Terminologi

Pada subbab ini penulis menjelaskan arti dari simbol-simbol yang akan digunakan pada subbab-subbab berikutnya. Secara umum, variabel diawali dengan huruf kapital. Term dan predikat diawali dengan huruf non-kapital.

- *X* menyatakan sebuah variabel *X*.
- _ menyatakan sebuah variabel *anonymous*.
- *a* menyatakan sebuah atom *a*.
- *f/n* menyatakan sebuah simbol fungsi *p* dengan arity *n*.
- *p/m* menyatakan sebuah simbol predikat *p* dengan arity *m*.
- $\bot$ menyatakan predikat *false*.

- $\bot \leftarrow L_1, ..., L_m$ menyatakan sebuah *integrity constraint*.
- $\mathcal{F} = \langle \mathcal{P}, \mathcal{AB}, IC \rangle$ menyatakan *abductive framework* $\mathcal{F}$ dengan $\mathcal{AB}$ merupakan himpunan predikat *abducible*, $\mathcal{P}$ merupakan sebuah program logika sedemikian sehingga tidak terdapat *rule* di $\mathcal{P}$ dengan *head* yang dibentuk oleh predikat pada $\mathcal{AB}$, dan IC merupakan himpunan *integrity constraint*.
- P menyatakan sebuah program input P.

3.3 Transformasi

Proses transformasi program input menjadi program output dilakukan oleh predikat *transform/*1. Predikat *transform/*1 akan melakukan transformasi pada program input P, menghasilkan program output P'. Berikut ini definisi predikat *transform/*1:

```
transform(Filename) :-
    clear,
    see_input_file(Filename),
    tell_output_file(Filename),
    transform_input_program,
    seen,
    told.
```

Kode 3.1: Definisi predikat *transform*(*Filename*)

Predikat *transform/*1 memiliki 6 buah predikat sebagai *goal-goal*-nya. Predikat *clear/*0 menghapus dan mendefinisikan ulang seluruh informasi yang mungkin tersisa dari proses transformasi sebelumnya. Informasi-informasi yang perlu dihapus dan didefinisikan ulang yaitu predikat-predikat dinamis yang tersimpan serta *trie* dan *table* yang sudah terbuat. Selanjutnya, predikat *see_input_file/*1 dan *see_output_file/*1 menentukan (membuka) input dan output *stream* selama proses transformasi, yaitu menjadikan berkas *Filename.ab* sebagai input *stream* untuk dibaca dan berkas *Filename.P* sebagai output *stream* untuk dituliskan. Proses transformasi itu sendiri dibungkus dalam predikat *transform_input_program/*0. Setelah input program P selesai ditransformasikan menjadi output program P', stream yang sudah dibuka sebelumnya ditutup kembali oleh predikat *seen/*0 dan *told/*0.

Berikut ini definisi predikat *transform_input_program/*0:

```
transform_input_program :-
    load_rules,
    add_indices,
    transform_per_rule,
    transform_if_no_ic,
    transform_abducibles.
```

Kode 3.2: Definisi predikat *transform_input_program/*0

Predikat *transform_input_program/*1 memiliki 5 buah predikat sebagai *goal-goal*-nya. Predikat *load_rules/*0 membaca program input *P* baris demi baris kemudian mengelompokkan sekaligus menyimpan *rule-rule* yang terdapat pada *P* dalam bentuk predikat dinamis *has_rule/*1 dan *rule/*2. Predikat *has_rule/*1 memiliki sebuah argumen *R*, menyatakan bahwa pada *P* terdapat definisi/*rule* mengenai *R*. Predikat *rule/*2 menyatakan definisi mengenai sebuah *rule*. Argumen pertama dan kedua dari *rule/*2 berturut-turut menyatakan *head* dan *body* dari *rule* tersebut. Selain itu, *load_rules/*0 juga sekaligus melakukan transformasi terhadap predikat-predikat pada *P* yang tidak memiliki definisi/*rule*, dengan kata lain, predikat-predikat yang hanya berupa fakta saja. Selanjutnya, predikat *add_indices/*0 mengubah setiap predikat dinamis *rule/*2 yang sudah tersimpan menjadi predikat dinamis yang baru *rule/*3. Predikat *rule/*3 menyatakan definisi mengenai sebuah *rule* beserta urutan pendefinisiannya pada *P*. Sebagai contoh, jika pada *P* terdapat *N* buah definisi/*rule* berbeda mengenai *Head*, maka akan terdapat *N* buah predikat dinamis *rule/*3: *rule*(*Head*, $Body_1$, *1*) yang menyatakan *rule* pertama mengenai *Head*, *rule*(*Head*, $Body_2$, *2*) yang menyatakan *rule* kedua mengenai *Head*, dan seterusnya hingga *rule*(*Head*, $Body_N$, *N*) yang menyatakan *rule* ke-*N* mengenai *Head*. Tiga predikat berikutnya yaitu *transform_per_rule/*0, *transform_if_no_ic/*0, dan *transform_abducibles/*0 merupakan predikat yang menangani proses transformasi. Detil mengenai ketiga predikat ini akan dijelaskan satu persatu.

Pertama, penulis akan menjelaskan detil dari predikat *transform_per_rule/*0. Berikut ini definisi predikat *transform_per_rule/*0:

```
transform_per_rule :-
    retract(has_rules(H)),
    find_rules(H, R),
    generate_apostrophe_rules(R),
    generate_positive_rules(H),
    generate_negative_rules(H, R),
```

```
    transform_per_rule.
```

Kode 3.3: Definisi predikat *transform_per_rule*/0

Predikat *transform_per_rule*/0 memiliki 6 buah predikat sebagai *goal-goal*nya. Predikat *retract*/1 menghapus sebuah predikat dinamis *has_rules*(H) yang tersimpan pada *database*. Informasi mengenai *rule* yang dihapus, yaitu rule mengenai H, digunakan oleh predikat *find_rules*/2 untuk menghasilkan R, yaitu semua *rule* mengenai H. Selanjutnya, H dan R digunakan oleh predikat *generate_apostrophe_rules*/1, *generate_positive_rules*/1, dan *generate_negative_rules*/2 untuk membentuk berturut-turut $\tau'(P)$, $\tau^+(P)$, dan $\tau^-(P)$ dan $\tau^*(P)$. *Goal* terakhir yaitu predikat *transform_per_rule*/0 merupakan pemanggilan rekursif yang akan berakhir ketika sudah tidak terdapat lagi predikat dinamis *has_rules*(H) pada *database*.

Kedua, penulis akan menjelaskan detil dari predikat *transform_if_no_ic*/0. Berikut ini definisi predikat *transform_if_no_ic*/0:

```
transform_if_no_ic :-
    find_rules(false, R),
    length(R, 0),
    generate_negative_rule_no_ic(NF).
transform_if_no_ic.
```

Kode 3.4: Definisi predikat *transform_if_no_ic*/0

Predikat *transform_if_no_ic*/0 akan membentuk $\tau^-(P)$ jika ditemukan bahwa $IC = \emptyset$. Untuk melakukannya, predikat *transform_per_rule*/0 membutuhkan 2 definisi. Pada definisi yang pertama, *transform_if_no_ic*/0 memiliki 3 buah predikat sebagai *goal-goal*nya. Predikat *find_rules*/2 menghasilkan R, yaitu semua *rule* mengenai $\perp$. Dengan kata lain, *find_rules*/2 menghasilkan himpunan *integrity_constraint* pada P, yaitu IC yang di-*unify* dengan R. Pada implementasi yang dibuat, himpunan *integrity_constraint* direpresentasikan sebagai sebuah *list*. Predikat *length*/2 merupakan predikat *built-in* yang argumen pertama dan keduanya berturut-turut adalah sebuah *list* dan sebuah bilangan bulat N, bernilai benar jika N merupakan panjang *list* tersebut. Pada predikat *transform_if_no_ic*/0, *length*/2 melakukan pengecekan apakah $R = \emptyset$, yaitu apakah $IC = \emptyset$. Jika ya, maka predikat berikutnya yaitu *generate_negative_rule_no_ic*/0 akan membentuk $\tau^-(P)$: $not\_\perp(I,I)$ yang di-*unify* dengan *NF*. Jika tidak, maka *length*/2 gagal dan beralih ke definisi *transform_if_no_ic*/0 yang kedua dan selalu sukses.

3.4 Implementasi *Cluster*

3.4.1 Instalasi *Frontend*

Tabel model lain, ditunjukkan pada tabel 3.1.

Tabel 3.1: Informasi *cluster* X

Host Name	X
Cluster Name	X
Certificate Organization	UI
Certificate Locality	Depok
Certificate State	West Java
Certificate Country	ID
Contact	X
URL	http://grid.ui.ac.id

Ada pagebreak disini.

Another type of table

Tabel 3.2: Perbandingan Partisi *default* dan manual

	Partisi default	Partisi manual yang dilakukan
/	16 GB	30 GB
/var	4 GB	18 GB
swap	1 GB	2 GB
/export	55 GB	26 GB

Program menghasilkan keluaran seperti pada kode 3.5.

Kode 3.5: Keluaran output

```
[root@nas-0-0 ~]# cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 sda4[0] sdb2[1]
      1917672312 blocks super 1.2 [2/2] [UU]

unused devices: <none>
[root@nas-0-0 ~]# mdadm --detail /dev/md0
/dev/md0:
        Version : 1.2
  Creation Time : Fri May  3 15:38:52 2013
     Raid Level : raid1
     Array Size : 1917672312 (1828.83 GiB 1963.70 GB)
  Used Dev Size : 1917672312 (1828.83 GiB 1963.70 GB)
   Raid Devices : 2
  Total Devices : 2
    Persistence : Superblock is persistent

    Update Time : Tue May 28 11:27:49 2013
          State : clean
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

           Name : nas-0-0.local:0  (local to host nas-0-0.local)
           UUID : 0754726d:3dfbd4b9:42b0f587:68631556
         Events : 28

    Number   Major   Minor   RaidDevice State
       0       8        4        0      active sync   /dev/sda4
       1       8       18        1      active sync   /dev/sdb2
```

3.4.2 Konfigurasi

Contoh verbatim dalam itemize :

- **Bold ini**
  dijalankan perintah berikut :

```
# javac Ganteng.java
# java Ganteng
```

  Perilaku sistem

```
# hai
# enable
# cd /export/rocks/install/
# create distro
# sh sesuatu.sh
# reboot
```

- **Menambahkan *package* pada *compute node***
  Langkah yang dilakukan adalah sebagai berikut :

  1. Masuk ke dalam direktori `/procfs/`
  2. Membuat/Mengubah berkas `xx.xml`. Jika tidak terdapat berkas tersebut, dapat disalin dari `skeleton.xml`.
  3. Menambahkan *package* yang ingin dipasang pada *compute node* diantara *tag* `<package>` seperti berikut : `<package>[package yang akan dipasang]</package>`.
  4. Menjalankan perintah berikut termasuk perintah untuk melakukan instalasi ulang seluruh *compute node*:

```
# cd /export/somedir
# create
# run host
```

3.4.2.1 semakin ke dalam

Kode 3.6: Keluaran mentah untuk detail *job*

```
[ardhi@xx ~]$ qstat -f 138
Job Id: 138.xx
    Job_Name = cur-1000-1np
    Job_Owner = ardhi@xx
    resources_used.cput = 27:21:35
    resources_used.mem = 86060kb
    resources_used.vmem = 170440kb
    resources_used.walltime = 27:24:50
    job_state = R
    queue = default
    server = hastinapura.grid.ui.ac.id
    Checkpoint = u
    ctime = Fri May 31 10:27:37 2013
    Error_Path = xx:/home/ardhi/xx/curcumin-1000/cur-1000-1np.e138
    exec_host = compute-0-5/0
    exec_port = 15003
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = e
    Mail_Users = ardhi.putra@ui.ac.id
    mtime = Fri May 31 10:27:47 2013
    Output_Path = xx:/home/ardhi/xx/curcumin-1000/cur-1000-1np.o138
    Priority = 0
    qtime = Fri May 31 10:27:37 2013
    Rerunable = True
    Resource_List.nodes = 1:ppn=1
    session_id = 5768
    etime = Fri May 31 10:27:37 2013
    submit_args = cur-1000-1np.pbs
    start_time = Fri May 31 10:27:47 2013
    submit_host = xx
    init_work_dir = /home/ardhi/xx/curcumin-1000
```

3.5 Pengujian

3.5.1 Kasus Uji

Berwarna!

Kode 3.7: Potongan skrip submisi *job* melalui torqace

```
# Go To working directory
cd $PBS_O_WORKDIR

#openMPI prerequisite
. /opt/torque/etc/openmpi-setup.sh
```

```
mpirun -np 5 -machinefile $PBS_NODEFILE mdrun -v -s \
  curcum400ps.tpr -o md_prod_curcum400_5np.trr -c lox_pr.gro
...
```

3.5.2 Kasus Uji

Contoh skrip yang dimasukkan pada *form* yang disediakan dapat dilihat pada kode 3.8.

Kode 3.8: Potongan `Makefile` *project*

```
# Make file for MPI
SHELL=/bin/sh

# Compiler to use
# You may need to change CC to something like CC=mpiCC
# openmpi : mpiCC
# mpich2  : /opt/mpich2/gnu/bin/mpicxx
CC=mpiCC
...
...
```

BAB 4
EVALUASI DAN ANALISIS

4.1 Hasil Pengujian

4.1.1 Hasil Pengujian Kasus Uji 1

Tabel lain. Hasil tersebut dapat dilihat pada tabel 4.1.

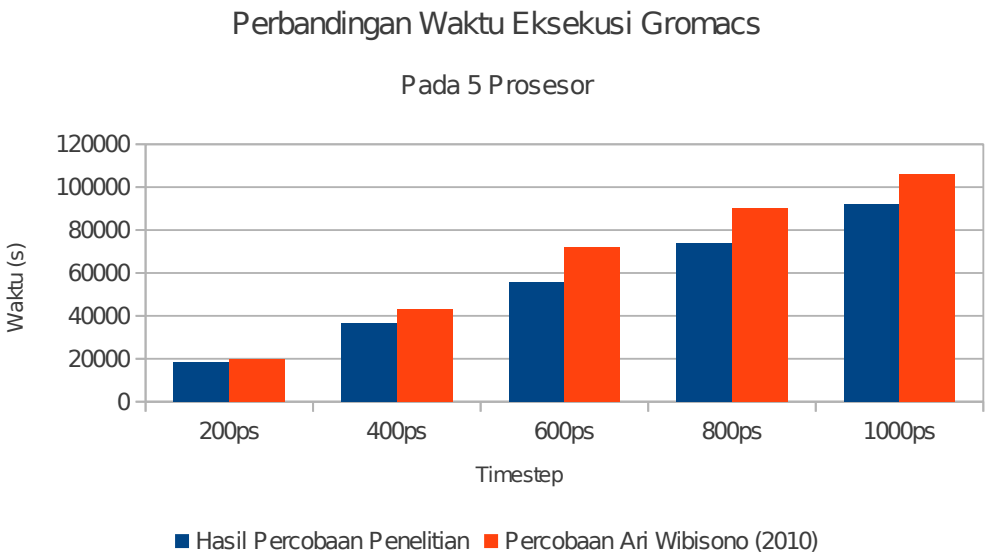
Tabel 4.1: Hasil pengujian menggunakan gromacs

No	*Timestep*	Waktu eksekusi berdasar jumlah prosesor		
		1	2	5
1	200ps	20h:27m:16s	12h:59m:04s	5h:07m:03s
2	400ps	1d:22h:40m:03s	1d:02h:08m:47s	10h:09m:39s
3	600ps	2d:23h:29m:21s	1d:14h:52m:52s	15h:25m:22s
4	800ps	4d:02h:05m:57s	2d:03h:30m:07s	20h:29m:38s
5	1000ps	5d:03h:29m:12s	2d:16h:32m:22s	1d:01h:34m:38s

4.2 Evaluasi Hasil Kasus Uji

4.2.1 Evaluasi Kasus Uji 1

Tabel 4.1 menunjukkan hasil uji coba pada penelitian ini. Gambar 4.1 menunjukkan perbandingan waktu eksekusi pada aplikasi x dengan jumlah prosesor sebanyak 5 buah.



Gambar 4.1: Perbandingan waktu eksekusi x untuk 5 prosesor

BAB 5
PENUTUP

Pada bab terakhir ini,

5.1 Kesimpulan

5.2 Saran

DAFTAR REFERENSI

Buyya, R. (1999). *High Performance Cluster Computing: Architectures and systems*. High Performance Cluster Computing. Prentice Hall PTR.

Guarddin, G. (2010). Percobaan Kompresi Menggunakan MPIBZIP2 pada Cluster Hastinapura. Peronal Communication.

Jackson, D. B., Snell, Q., dan Clement, M. J. (2001). Core algorithms of the maui scheduler. In *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '01, pages 87–102, London, UK, UK. Springer-Verlag.

Jones, J. P., Lifka, D., Nitzberg, B., dan Tannenbaum, T. (2002). Cluster workload management. In Sterling, T., editor, *Beowulf cluster computing with Linux*, pages 301–306. MIT Press, Cambridge, MA, USA.

McGuire, T. J. (2010). A gentle way of introducing multi-core programming into the curriculum: tutorial presentation. *J. Comput. Sci. Coll.*, 26(1):124–125.

Mell, P. dan Grance, T. (2009). The NIST definition of cloud computing. Technical report, National Institute of Standards and Technology, Information Technology Laboratory. [Diakses 17 April 2013].

Mozdzynski, G. (2012). Concepts of Parallel Computing. http://www.ecmwf.int/services/computing/training/material/com_hpcf.html. [Diakses 20 Maret 2013].

Neal, R. (2010). Cloud computing brings savings in energy, maintenance costs. http://www.federaltimes.com/article/20100323/IT03/3230304/Cloud-computing-brings-savings-energy-maintenance-costs. [Diakses 1 April 2013].

Oxford Dictionaries (2010). http://oxforddictionaries.com/definition/english/parallel. [Diakses 1 April 2013].

Pressman, R. (2010). *Software engineering: a practitioner's approach*. McGraw-Hill higher education. McGraw-Hill Higher Education.

Treese, W. (2004). How to build a supercomputer. *netWorker*, 8(4):15–18.

LAMPIRAN

LAMPIRAN 1 : KODE SUMBER

admin_useraddmaster

Skrip ini diletakkan pada direktori `/usr/sesuatu` dan hanya dapat dieksekusi oleh *root*. Skrip ini berguna untuk menambahkan pengguna baru sesuai dengan konfigurasi baru yang telah ditetapkan.

Kode 1: Skrip menambahkan pengguna baru

```
#!/bin/csh -f
blah blah blah
blah blah blah
blah blah blah
blah blah blah
blah blah blah
```

getuser.cron

Penjelasan skrip disini

Kode 2: *Cronjob* menambahkan pengguna baru

```
#!/bin/bash
# Change these two lines to localize to your system:
# Adapted from /usr/local/sbin/admin_useradd

cat /dev/null > $userlist
for (( i=0; i<${#listemailto[@]}; i++ ))
do
        uname=${listusername[$i]}
        mailto=${listemailto[$i]}

        echo "User $uname created, please use torqace wisely." | mail -s "Torqace
             user registration" $mailto
done
```

LAMPIRAN 2 : BERKAS KONFIGURASI

compute.xml

Kode 3: Berkas `compute.xml`

```
<?xml version="1.0" standalone="no"?>
<kickstart>
<description>
  Compute node XML file
</description>
</kickstart>
```

LAMPIRAN 8 : UAT DAN KUESIONER

Tabel 1: Tabel UAT dan Kuesioner

No.	Langkah Penggunaan	Fitur Berjalan	Tingkat Kemudahan (1-5)	Tingkat Kepuasan (1-5)	Saran / Komentar
		Berhasil /Tidak	1:Sangat sulit ; 5:sangat mudah	1 : Sangat kecewa ; 5 : sangat puas	
Use Case : Login					
1.1	Pengguna berada pada halaman depan torqace				
1.2	Pengguna memasukkan username dan password pada field yang telah disedi-akan.Kemudian menekan tombol 'login'				
1.3	Apabila Sukses, maka pengguna masuk ke dalam sistem dan dihadapkan pada menu utama				
Use Case : Register					
2.1	Pengguna berada pada halaman registrasi pengguna torqace				

2.2	Pengguna memasukkan user-name,password, dan email pada field yang telah disediakan. Kemudian menekan tombol 'submit'				
2.3	Sistem akan mengonfirmasi masukan, dan akan mengirimkan email untuk mem-beritahu pengguna apabila proses pendaf-taran telah selesai				
Use Case : Logout					
3.1	Pengguna memilih menu untuk melakukan logout				
3.2	Sistem akan mengeluarkan pengguna, dan pengguna tidak dapat menggunakan fitur-fitur utama aplikasi				
Use Case : Upload Job Sederhana					
4.1	Pengguna memilih menu upload file/pro-ject pada menu utama				
4.2	Pengguna memilih pilihan 'single file' pada tipe project				

4.3	Pengguna memilih berkas yang akan diunggah, mengisi label, dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
4.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				
4.5	Sistem akan menampilkan informasi terkait berkas yang diupload				
Use Case : Upload Job Compressed					
5.1	Pengguna memilih menu upload file/project pada menu utama				
5.2	Pengguna memilih pilihan 'compressed files' pada tipe project				
5.3	Pengguna memilih arsip yang akan diunggah, mengisi label, menentukan akan melakukan make atau tidak dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
5.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				

5.5	Sistem akan menampilkan informasi terkait berkas yang diupload dan diekstrak. Keluaran make juga akan ditampilkan bila dipilih				
Use Case : Upload Array Job					
6.1	Pengguna memilih menu upload file/project pada menu utama				
6.2	Pengguna memilih pilihan 'array' pada tipe project				
6.3	Pengguna memilih arsip-arsip yang akan diunggah, mengisi label, menentukan akan melakukan make atau tidak dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
6.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				
6.5	Sistem akan menampilkan informasi terkait berkas yang diupload dan diekstrak. Keluaran make juga akan ditampilkan bila dipilih				

Use Case : Melihat antrian pada queue					
7.1	Pengguna memilih menu queue status pada menu utama				
7.2	Pengguna berada pada halaman yang berisi informasi queue				
Use Case : Melihat detil antrian					
8.1	Dari halaman status queue, pengguna memilih job tertentu				
8.2	Informasi mengenai detil job tersebut ditampilkan dalam bentuk tabel				
8.2.1	Apabila job tersebut bukan milik pengguna, maka sistem akan melarang pengguna melihat informasi detil suatu job				
Use Case : Membuat script job					
9.1	Pengguna memilih untuk melakukan 'generate script' baik dari laporan upload berkas, atau dari penjelajahan direktori				
9.2	Pengguna mengisi nama job, parameter job, dan script yang akan dijalankan.				
9.3	Pengguna mengonfirmasi konfirmasi submit job				

9.4	Pengguna dapat melihat informasi script secara keseluruhan dan pesan apakah terjadi kegagalan atau tidak, serta id job yang diberikan				
Use Case : Load spesifikasi job lain					
10.1	Pengguna berada pada halaman untuk membuat script				
10.2	Pengguna memilih 'Load a Previous Job'				
10.3	Pengguna memilih job mana yang akan dimuat dan menekan tombol 'Load'				
10.4	Pengguna kembali ke halaman pembuatan script dengan spesifikasi job sebelumnya				
Use Case : Menjelajah Direktori					
11.1	Pengguna memilih menu 'View File/Project' pada menu utama				
11.2	Pengguna dapat melakukan navigasi untuk masuk ke dalam direktori tertentu, atau kembali ke direktori diatasnya, dan dapat melihat terdapat berkas apa saja dalam direktori				

Use Case : Menghapus Berkas/Direktori					
12.1	Pengguna berada pada halaman penjelajahan direktori				
12.2	Pengguna memilih pilihan untuk menghapus berkas/direktori di samping item yang akan dihapus				
12.3	Pengguna mengonfirmasi konfirmasi penghapusan				
Use Case : Mengunduh Berkas/Direktori					
13.1	Pengguna berada pada halaman penjelajahan direktori				
13.2	Pengguna memilih pilihan untuk mengunduh berkas/direktori di samping item yang akan dihapus				
Use Case : Melihat Berkas					
14.1	Pengguna berada pada halaman penjelajahan direktori				
14.2	Pengguna memilih berkas yang berupa berkas teks				
14.3	Sistem akan menampilkan konten dari berkas tersebut				