Penggunaan Algoritma Branch and Bound dalam Parallel Applications

Andika Kurniawan Susilo
13506104

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institute Teknologi Bandung
Jl.Ganesha 10, Bandung

If16104@students.if.itb.ac.id

ABSTRAK

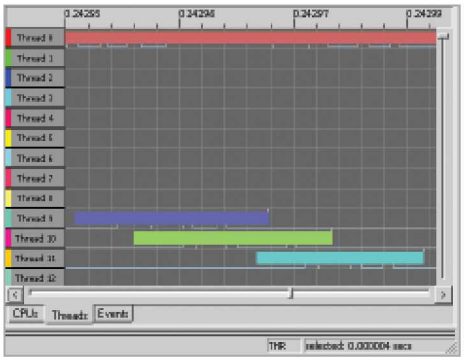
Parallel Applications* bekerja sangat dinamis dalam hal komputasi dan komunikasi sehingga sangat sering terjadi perubahan ketika komputer sedang berjalan. Contoh paling konkrit *Parallel Applications* adalah *Internet,* setiap saat komputer dapat mengirim atau menerima data dari beberapa komputer atau lebih. Jika tidak ada suatu program yang mengaturnya pasti akan terjadi kekacauan. Sehingga dibutuhkan suatu tehnik agar tidak terjadi *failover* (keadaan dimana sebuah *application* menunggu *application* yang lain). *Load balancing* digunakan untuk mengatasi permasalahan ini. Load balancing digunakan untuk mengatur *mapping* objek *ke processor*. Dalam kasus ini akan menjadi lebih ekonomis untuk penghematan waktu dan optimasi dalam proses *mapping* sebuah objek ke *processor* dengan menggunakan strategi Algoritma *branch and bound.

Kata kunci: Parallel Applications, Branch and Bound , Load Balancing , Load Balancer.

1. PENDAHULUAN

Pembangunan *Parallel Applications* yang efisien menjadi sulit ketika aplikasi dapat menjadi dinamis, kadangkala diam dan juga bisa meminta pemrosesan data dalam jumlah yang banyak. Dalam kasus tertentu terjadi masalah terhadap kinerja dari komputer ketika terjadi *Load Imbalances*, atau bisa dikatakan *failover*. Dalam keadaan *imbalance* aplikasi/link yang menunggu aplikasi/link yang lain dapat dieksekusi jika aplikasi yang ditunggu sudah selesai digunakan atau mengalami *fail* (kegagalan mengeksekusi). Meskipun imbalances yang terjadi biasanya kecil dan dapat ditoleransi ketika sedang menjalankan jumlah kecil *processor*. tetapi akan menjadi masalah utama dalam sistem ketika menggunakan *processor* dalam jumlah besar.

Untuk Mengatasi hal tersebut dapat digunakan Algoritma *branch and bound*. Dengan menganggap aplikasi atau link sebagai objek yang akan dimasukkan ke dalam *searching tree* dan ditentukan nilai *cost*nya.



Gambar 1 ***Parallel Execution***

2. APLIKASI ALGORITMA

Bab ini akan membahas bagaimana pendekatan algoritma *branch and bound* yang akan digunakan oleh *load balancing.* Pertama-tama semua aplikasi paralel akan dimodelkan sebagai objek yang . Nilai *cost* dari objek dimodelkan berdasarkan karakteristik dari setiap mesin itu sendiri dan khusus untuk petukaran data dalam satu link atau aplikasi nilai *cost*nya akan dihiraukan.*load* balancer

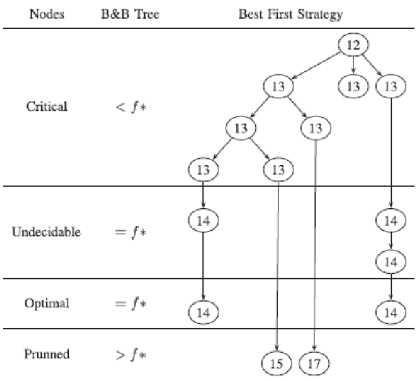
bebas menentukan penggunaan objek ke setiap *processor* untuk memaksimalkan kinerja program.

Objek yang akan diseimbangkan akan direpresentsikan sebagai jaringan komunikasi antar *entity*. Objek-objek akan direpresentasikan dalam *graf* G = (jumlah message, total bytes yang dikirim). Dengan *source* adalah sumber pengirim dan *sink* adalah tempat yang dikirim.

Berikut adalah cara untuk menghitung *cost*,

- G(N,E) dengan
  - N = N1 U … U Np, dengan setiap |Ni| ~ |N|/p
  - beberapa *edges* terhubung ke Ni and Nk yang memungkinkan.
- Jika N = {*tasks*}, setiap *unit cost*, edge e=(i,j) berarti setiap *task* I harus berkomunikasi dengan *task*
  - *balancing the load*, i.e. untuk setiap |Ni| ~ |N|/p
  - *minimizing communication*

Objek akan ditempatkan dalam *decrease sequence* berdasarkan *cost* dari *assignment*. Lalu objek yang lebih bayak nilai *cost*nya akan ditempatkan ke level yang lebih tinggi dari *search tree*.



Gambar 2 *node classification* untuk *search tree*

Setelah semua cost dimasukkan ke dalam *searching tree* ,

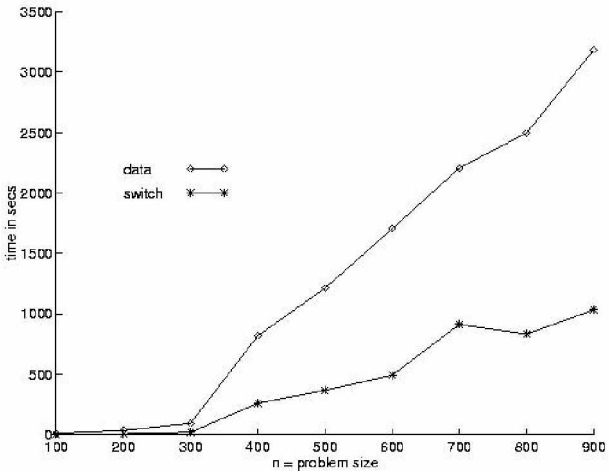
```
Node root; // the root node.
PriorityQueue set; // the priority queue that  stores the
                   // set of pendings nodes
Node incumbent; // the Incubent.

 set.Ins(root);
 while (! set.empty() ) {
    Node n =  set.deletemin();
    foreach ( Node s son of n) { // Branching
        s.Evaluate(); // Evaluation of the node s
        if ( s.IsSolution() && incumbent.Cost()>s.Cost() ){
              incumbent= son;
      pq.Deletegreater(incumbent.Cost());
        }
```

```
            else if ( s.Feasible() && incumbent.Cost()>s.Eval() )
                  set.Insert(s);
        }
    }
```

untuk algoritma eksekusinya seperti di bawah ini

Grafik dibawah ini menunjukkan nilai waktu yang dibutuhkan oleh *load balancer* untuk mengeksekusi n buah masalah.



Gambar 3 grafik antara jumlah n dan waktu yang dibutuhkan

3. PERBANDINGAN PERFORMANSI DENGAN ALGORITMA LAIN

Dalam bab ini akan membandingkan Algoritma *branch and bound* untuk *load balancer* dengan empat algoritma yang lain, yaitu

- *Greedy* :
  Algoritma ini menggunakan prinsip *greedy*
- *Random* :
  Objek secara acak didistribusikan antar *processor*.
- *Greedy – refine* :
  Algoritma *Greedy* dijalankan untuk mendistribusikan *processor*, prosedure refinement ini melihat setiap *processor* dengan *load* yang berada di bawah rata-rata dari setiap *treshold*, dan memindahkannya ke *under-loaded processors*, sampai tidak ada lagi perpindahan yang memungkinkan.
- *Random-Refine* :
  Prosedur *refinement* diaplikasikan kedalan solusi yang berasal dari proses *Random* algoritma.

Case #	Procs.	Comm. Cost	Greedy	Greedy-Refine	Random	Random-Refine	Branch & Bound
1	9	0	99.7	99.7	69.1	69.1	99.8
2	20	0	98.4	98.4	57.5	57.5	99.4
3	9	120	51.4	55.6	58.5	68.6	81.0
4	20	120	28.8	31.7	50.6	67.7	78.4
5	9	250	34.4	37.0	48.4	55.9	64.4
6	20	250	26.3	28.5	41.2	44.7	60.1
7	9	300	37.1	40.9	46.0	50.9	60.3
8	20	300	26.7	30.0	39.1	42.1	56.2
9	9	400	44.2	52.2	41.8	50.5	54.6
10	20	400	21.2	24.0	35.4	36.9	49.6
11	9	500	26.9	28.9	38.4	46.4	49.5
12	20	500	27.4	30.0	32.3	42.3	43.7
13	9	600	29.9	34.7	35.4	41.7	44.3
14	20	600	13.6	14.2	29.6	38.0	39.5
15	9	700	20.9	22.2	32.9	38.4	41.1

tabel 1 menampilkan hasil dari kelima algoritma ketika diterapkan dalam program dan dijalankan

Tsend = _send _ Nmessages + _send _ Nbytes

Treceive = _receive _ Nmessages + _receive _ Nbytes

Dalam semua kasus, efisiensi dihitung dengan cara *Tsequential=(P _ Tparallel)*, dimana P adalah jumlah *processor*. Tabel di atas menjelaskan ketika program dijalankan dengan waktu yang terbatas, *algoritma branch and bound* memberikan solusi paling efisien diantara algoritma lain yang diimplementasikan.

Berdasarkan hasil yang didapat , dapat disimpulkan bahwa efisiensi dari solusi disetiap algoritma menurun saat komunikasi overhead meningkat. hal ini disebabkan oleh optimal efisiensi itu sendiri yang akan menurun ketika komunikasi *overhead* naik.

Selain itu juga dimonitor kualitas dari solusi yang dihasilkan dari fungsi waktu yang digunakan oleh *load balancer*. seperti yang diharapkan kualitas meningkat dengan lebih banya search.

Dapat disimpulkan bahwa dengan menggunakan *refinement* algoritma tidak meningkatkan waktu yang disimpan secara memuaskan oleh satupun dari algoritma *load balancing* , tetapi dalam hal lain dapat menghasilkan solusi yang lebih baik dalam banyak kasus. Contohnya, dalan kasus *refine* diaplikasikan untuk *Greedy* membutuhkan waktu 1 detik lebih lama, dan 10 persen kenaikan efisiensi. Untuk algoritma *random*, r*efinement* membutuhkan proporsi lebih banyak waktu, tetapi efisiensi yang dihasilkan meningkat secara dramatis.

4. KESIMPULAN

1. Parallel Aplication yang bekerja sangat dinamis, sehingga dibutuhkan strategi load balancing.
2. Algoritma *Branch and Bound* dapat digunakan dalam mengoptimalisasikan kinerja *Parallel Aplication* dalam hal mapping objek kedalam *processor*.
3. Kegunaan dari algoritma *Branch and Bound* adalah kemampuannya menggunakan waktu yang diberikan untuk menghasilkan *mapping* yang terbaik.
4. *Intellegent Greedy* dapat juga diterapkan untuk *Parallel Aplikation* tetapi tidak seefektif dari *Branch and Bound* .

REFERENSI

[1] Munir,Rinaldi."Strategi Algoritmik",Lab Ilmu dan Rekayasa Komputasi, Departemen Teknik Informatika ITB.

[2] Radhakrishnan, Shobana,dkkl," Branch and Bound Based Load Balancing for Parallel Applications", University of Illinois at Urbana-Champaign, Urbana, IL, USA

[3] Lau,K.K dan Kumar, M.J, "Parallel Implementation of Branch and Bound Algorithm for Solving Vehicle Routing Problem on NOWs" , Department of Mathematics and Statistics, Cur-tin University of Technology Australia

[4] Moe1, Randi dan Sørevik2,Tor, "Parallel Branch and Bound Algorithms on Internet Connected Workstations, Dept. of Informatics, University of Bergen, Norway

[5] http://en.wikipedia.org/wiki/Path_
Tanggal Akses : 20 Mei 2008 jam 03.00

[6] http://www.webopedia.com/TERM/L/device.html
Tanggal Akses : 20 Mei 2008 jam 03.05

[7] http://www.pc24.co.id/article/category40_1.htm
Tanggal Akses : 20 Mei 2008 jam 03.10