ISSN 2355-3286

# Proxy Selector Berbasis Link-State

Edwin Tunggawan[1], Hargyo Tri Nugroho I.[2]
[1]Program Studi Teknik Informatika, Universitas Multimedia Nusantara, Tangerang, Indonesia
vcc.edwint@gmail.com, [2] hargyo@umn.ac.id
[2]Program Studi Sistem Komputer, Universitas Multimedia Nusantara, Tangerang, Indonesia
hargyo@umn.ac.id

Diterima 2 Mei 2013
Disetujui 16 Mei 2013

**Abstrak—**Pemilihan *proxy server* yang digunakan untuk menjelajah web seringkali dilakukan secara manual oleh pengguna. Proses ini dapat diotomatisasi dengan menggunakan aplikasi *proxy selector*. Pada penelitian ini, *proxy selector* dikembangkan dengan mengadaptasi algoritma *link-state*. Algoritma disesuaikan untuk pemilihan proxy. Aplikasi juga diberikan kemampuan untuk melakukan prediksi dengan menggunakan perhitungan *moving average* untuk menghitung perkiraan performa *proxy server* yang diuji. Untuk mendapatkan hasil perhitungan yang lebih akurat, diilakukan percobaan untuk menentukan panjang periode (*n*) *moving average* yang paling baik dengan menggunakan perhitungan *mean squared error* terhadap data. Dari hasil perhitungan tersebut, didapatkan panjang periode yang memberikan hasil prediksi terbaik pada data percobaan.

**Keywords—***proxy selector*, *link-state*, *moving average*, *mean squared error*

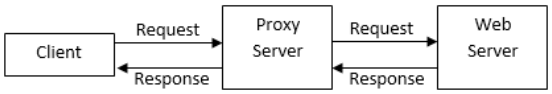
## I. Pendahuluan

Pada saat ini, Internet telah digunakan secara luas untuk memenuhi berbagai kebutuhan Informasi masyarakat. Menjelajah web merupakan salah satu aktivitas yang paling banyak dilakukan oleh pengguna Internet. Pada saat mengakses suatu situs web, pengguna menggunakan sebuah *client host* yang akan mengakses sebuah *web server*. *Web server* kemudian akan memberikan respon berupa informasi yang diinginkan oleh pengguna. Respon akan dikirim ke *client*.

Ada kalanya dibutuhkan sebuah *proxy server* sebagai penengah di antara *client* dan *web server*. *Proxy server* memiliki beberapa manfaat bagi pengguna, antara lain untuk mempercepat akses ke laman web dan anonimitas.

Dalam sebuah instansi yang memiliki jaringan komputer cukup besar, *proxy server* dapat digunakan untuk mempercepat akses pengguna jaringan tersebut ke laman web dengan mengaktifkan fungsi *caching* pada *proxy server*. Dengan menyimpan *cache* dari laman web, *proxy server* dapat mengambil *cache* laman web yang diminta oleh *client* dan mengirimkannya ke *client* yang bersangkutan. *Request* tidak perlu selalu diteruskan ke *web server* karena *proxy server* menyimpan data dari laman web tersebut di dalam *cache* untuk diberikan ke *client* pada *request* selanjutnya. Dengan demikian, kecepatan akses ke laman web itu akan meningkat.

Terkadang pengguna perlu mengakses suatu laman web tanpa memberikan informasi mengenai lokasi dan *host* yang sedang digunakan ke *web server*. Pada situasi ini, pengguna dapat menggunakan *proxy server* sebagai perantara untuk mengakses laman web tersebut. *Request* dari *client* akan diteruskan oleh *proxy server* ke *web server*. Dalam situasi ini, yang akan tercatat sebagai *host* pengakses laman web di *web server* bukanlah *client*, melainkan *proxy server* yang menjadi perantara. Karena itu, *web server* tidak mendapatkan informasi mengenai *client* dan anonimitas pengguna tetap terjaga.




Gambar 1. *Proxy Server* Meneruskan *Request* dan *Response*

Untuk menggunakan *proxy server*, *client host* harus dikonfigurasi terlebih dahulu oleh pengguna agar mengakses laman web melalui *proxy server* yang dipilih. Dalam banyak kasus, terutama untuk mendapatkan anonimitas, pengguna harus memilih *proxy server* secara manual sebelum melakukan konfigurasi.

Penelitian ini mengadaptasikan algoritma *link-state* untuk melakukan pemilihan *proxy server*. Algoritma *link-state* umum digunakan oleh *router* untuk memilih rute terbaik dalam meneruskan data agar sampai ke *host* tujuan.

## II. Algoritma Link-State

Salah satu algoritma *link-state routing* yang paling banyak digunakan adalah OSPF (*Open Shortest Path First*) [1]. Berikut ini adalah langkah-langkah yang dilakukan untuk menentukan rute pada OSPF [2].

- Setiap *router* akan bertukar informasi dengan

*router-router* tetangganya.

- Setelah mendapatkan informasi *link-state*, *router* akan menghitung jarak dengan menggunakan algoritma Dijkstra.
- *Router* akan mengirimkan paket data ke *router* lainnya untuk mengecek apakah kondisi *link-state* sesuai dengan informasi yang dimiliki.
- Jika terdapat perbedaan, *router* akan melakukan *update* pada *routing table* dan menyebarkannya ke *router* lainnya.

*Moving Average dan Mean Squared Error*

*Moving average* digunakan untuk melakukan analisa terhadap kecenderungan suatu nilai untuk bertambah, berkurang, atau tetap. *Moving average* umum digunakan dalam analisa pasar dan saham, dengan membandingkan nilai *moving average* data yang didapatkan dari perusahaan A dan perusahaan B. Data dihitung dengan interval (*n*) yang sama, kemudian hasilnya dianalisa untuk menentukan prediksi [3]. Berkut ini adalah rumus *simple moving average*.

$$SMA = \frac{p_M + p_{M-1} + \cdots + p_{M-(n-1)}}{n}$$

Gambar 2. *Simple Moving Average*

Interval dari *moving average* berpengaruh terhadap seberapa baik hasil prediksi yang diberikan. Untuk mengukur seberapa baik hasil prediksi dari suatu perhitungan *moving average*, dapat digunakan perhitungan *mean squared error* (MSE). Berikut ini adalah rumus MSE.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2.$$

Gambar 3. *Mean Squared Error*

Prediksi yang paling baik akan menghasilkan nilai MSE yang paling rendah [4].

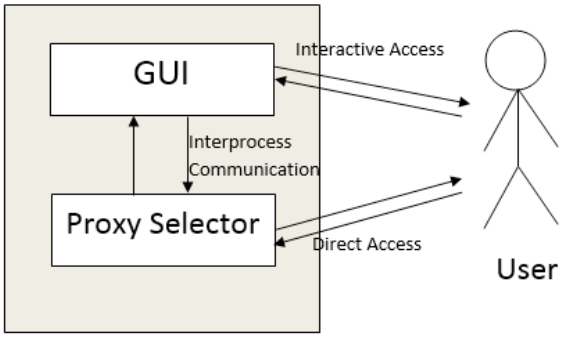
*Pemilihan Server dengan Pendekatan Probabilistik*

Dalam melakukan pemilihan *server* untuk keperluan *load-balancing*, pendekatan probabilistik dapat digunakan untuk menghitung kemungkinan-kemungkinan pemilihan *server* oleh *client* [5]. Perhitungan tersebut bermaksud untuk mengantisipasi agar tidak terlalu banyak *client* yang mengakses satu *server* secara bersamaan sehingga *server* tersebut mengalami *overload*.

Pendekatan probabilistik juga dapat digunakan dalam jaringan komunikasi P2P, agar *client* dapat secara otomatis memilih konten yang diperkirakan akan dibutuhkan untuk proses selanjutnya dari *peer* yang telah dipilih berdasarkan perkiraan kinerja dari *peer* yang bersangkutan. Perkiraan tersebut dibuat berdasarkan kinerjanya pada beberapa operasi terakhir [6].

## III. Struktur Aplikasi

Aplikasi dibagi menjadi dua bagian, yaitu *proxy selector* yang dituliskan dengan menggunakan bahasa pemrograman Python dan GUI yang dituliskan dengan menggunakan bahasa pemrograman C#. Pengguna dapat mengakses langsung program *proxy selector* yang berbasis *console* atau menggunakan GUI yang melakukan *interprocess communication* dengan *proxy selector*.




Gambar 4. Interaksi antara GUI, *Proxy Selector*, dan Pengguna

*Proxy server* dapat melakukan *caching* untuk halaman-halaman yang telah diakses, dan jika *client* me-*request* halaman tersebut maka *proxy server* akan memberikan data yang tersimpan di dalam *cache*. Jika *client* me-*request* halaman yang belum terdapat pada *cache*, maka *proxy server* akan meneruskan *request* ke *web server* untuk meminta halaman yang diinginkan, kemudian memberikannya pada *client*. Karena itu, pengujian untuk memilih *proxy server* dibagi menjadi dua modus: tanpa *cache* dan dengan *cache*.

Topologi *logic* pada sistem komunikasi antara *client*, *web server*, dan *proxy server* lebih sederhana jika dibandingkan dengan topologi sistem *routing*. Karena itu, algoritma *link-state* yang digunakan dalam aplikasi ini adalah bentuk yang telah disederhanakan dan diadaptasikan untuk sistem komunikasi *client*, *proxy server*, dan *web server*.

- *Proxy selector* mengecek daftar *proxy server* dari sebuah *text file*.
- Setelah memuat daftar *proxy server*, *proxy selector* akan mengecek *history* dari kondisi *link-state* dan prediksi yang telah didapatkan sebelumnya.
- *Proxy selector* melakukan pengujian performa kepada *proxy server* yang terdaftar, mengukur kecepatannya, dan menghitung prediksi performanya untuk beberapa saat ke depan dengan *moving average* dan mencatat hasilnya ke dalam *text file*.
- Program memberikan *output* berupa data *testing history* dan prediksi untuk dibaca oleh pengguna.

Pada poin ketiga dari empat poin di atas, ditambahkan perhitungan *moving average* untuk melakukan prediksi performa *proxy server* untuk beberapa waktu ke depan. Berbeda dengan sistem *routing* yang memiliki jumlah *node* lebih banyak untuk melakukan lompatan (*hop*), jumlah *node* pada topologi *logic* untuk sistem komunikasi *client*, *proxy server*, dan *web server* hanya memiliki satu *node* untuk melakukan lompatan. Tetapi kecepatan *node* tersebut cenderung tidak stabil jika dibandingkan dengan *node* pada sistem *routing* jika yang digunakan adalah *proxy server* yang alamatnya disebarkan di Internet. Karena itulah ditambahkan kemampuan untuk melakukan prediksi pada aplikasi ini.

## IV. Percobaan dan Analisa Data

Untuk mendapatkan data, dilakukan pengujian terhadap tiga buah *proxy server* berikut.

- 202.137.22.182:8080 (*Proxy 1*)
- 116.213.51.82:80 (*Proxy 2*)
- 118.98.35.251:8080 (*Proxy 3*)

Pengumpulan data dibagi menjadi dua bagian, yaitu pengumpulan data dengan memperhitungkan *cache* dan pengumpulan data tanpa memperhitungkan *cache*. Untuk kedua bagian pengumpulan data tersebut dilakukan empat kali percobaan terhadap *proxy server*. Pada masing-masing percobaan, *proxy selector* akan mengirimkan *request* sebanyak 60 kali dengan *timeout* 10 detik kepada ketiga *proxy server*.

Setelah data didapatkan, dilakukan perhitungan dengan menggunakan *moving average* dengan interval (*n*) dalam *range* 2-10 untuk setiap percobaan yang dilakukan. Berdasarkan hasil perhitungan *moving average* itu, dibuat perhitungan MSE untuk mencari panjang interval yang memberikan prediksi paling baik untuk data hasil percobaan.

*Tanpa Cache*

Berikut ini adalah nilai *n* yang memberikan nilai MSE minimum pada setiap percobaan pada pengumpulan data tanpa memperhitungkan *cache* dari *proxy server*.

Tabel 1. Interval dengan MSE Minimum

| Percobaan | *Proxy 1* | *Proxy 2* | *Proxy 3* |
|---|---|---|---|
| I | 10 | 10 | 10 |
| II | 10 | 10 | 10 |
| III | 10 | 10 | 10 |
| IV | 10 | 7 | 7 |

Pada tabel terlihat jika pada percobaan I sampai dengan percobaan III, prediksi terbaik pada semua *proxy server* didapatkan dengan menggunakan nilai *n* sebesar 10 pada perhitungan *moving average*. Pada percobaan IV, nilai 10 memberikan prediksi terbaik pada *Proxy 1* sementara untuk *Proxy 2* dan *Proxy 3* prediksi terbaik didapatkan dengan nilai *n* sebesar 7.

Dengan demikian, dari 12 kasus pada percobaan I sampai dengan percobaan IV, nilai *n* sebesar 10 memberikan prediksi terbaik pada 10 kasus, sementara nilai *n* sebesar 7 hanya memberikan prediksi terbaik pada 2 kasus. Maka nilai 10 dianggap sebagai panjang interval yang paling baik.

*Dengan Cache*

Berikut ini adalah nilai *n* yang memberikan nilai MSE minimum pada setiap percobaan pada pengumpulan data dengan memperhitungkan *cache* dari *proxy server*.

Tabel 2. Interval dengan MSE Minimum

| Percobaan | *Proxy 1* | *Proxy 2* | *Proxy 3* |
|---|---|---|---|
| I | 10 | 10 | 10 |
| II | 10 | 10 | 10 |
| III | 10 | 10 | 10 |
| IV | 10 | 7 | 7 |

Nilai *n* yang menghasilkan nilai MSE minimum dengan memperhitungkan *cache* ini lebih bervariasi dibandingkan perhitungan tanpa *cache*. Pada percobaan I, *Proxy 1*, *Proxy 2*, dan *Proxy 3* masing-masing memiliki hasil prediksi terbaik pada nilai *n* sebesar 7, 8, dan 9. Pada percobaan II, hasil prediksi terbaik untuk ketiga *proxy server* tersebut didapatkan pada nilai *n* sebesar 10, 7, dan 10. Untuk percobaan III, prediksi terbaiknya didapatkan pada nilai *n* sebesar 10, 9, dan 10, sementara untuk percobaan IV nilai *n* yang memberikan prediksi terbaiknya adalah sebesar 7, 10, dan 10.

Dengan demikian, dari 12 kasus pada percobaan I sampai dengan percobaan IV, nilai *n* sebesar 10 memberikan prediksi terbaik pada 6 kasus, sementara nilai *n* sebesar 9 pada 2 kasus, nilai *n* sebesar 8 pada 1 kasus, dan nilai *n* sebesar 7 memberikan prediksi terbaik pada 2 kasus. Karena lebih banyak memberikan hasil prediksi yang terbaik, nilai 10 masih dianggap sebagai panjang interval yang paling baik untuk perhitungan dengan *cache*.

## V. Uji Performa Aplikasi

Pengujian performa aplikasi dilakukan dengan menggunakan AMD CodeAnalyst pada komputer dengan *processor* Intel Core2 Duo T6600 dan RAM sebesar 2 GB. Penggunaan rata-rata RAM untuk aplikasi secara keseluruhan menurut AMD CodeAnalyst adalah 0,97% dari total RAM yang tersedia, yaitu sebesar 19,4 MB. Aplikasi GUI-nya sendiri berukuran 372 KB, dan terbagi menjadi dua bagian utama: *splash screen* dan *form* utama aplikasi.

Pada saat AMD CodeAnalyst digunakan hanya untuk menguji *splash screen*, penggunaan RAM dari aplikasi adalah 0% dari total RAM yang tersedia. Penggunaan RAM sebesar 0 B untuk menjalankan aplikasi adalah tidak mungkin, karena aplikasi tersebut harus dimuat di RAM. Jika dilakukan perhitungan, persentase ukuran aplikasi dari total RAM adalah (372.000 / 2.000.000) x 100% = 0,000186%. Nilai tersebut cukup dekat dengan nilai 0% yang diberikan

oleh AMD CodeAnalyst, sehingga dapat disimpulkan jika penggunaan RAM oleh aplikasi sejauh ini adalah normal.

Saat AMD CodeAnalyst digunakan untuk menguji *form* utama aplikasi dan menjalankan *proxy selector* dengan GUI selama kurang lebih 5 menit, AMD CodeAnalyst memberikan hasil penggunaan RAM sebesar 0,98% dari total RAM yang tersedia, yaitu sebesar 19,6 MB. Nilai ini jauh lebih besar dibandingkan konsumsi sumber daya RAM oleh *splash screen* karena pada *form* utama terdapat lebih banyak fungsi, variabel, *class*, dan *object* yang digunakan untuk operasi dan lebih banyak *dynamic-linking library* yang ikut dimuat ke RAM.

Jika *proxy selector* dijalankan tanpa GUI, AMD CodeAnalyst memberikan hasil penggunaan RAM sebesar 0% dari total RAM yang tersedia. *Proxy selector* tanpa GUI ini adalah sebuah *script* Python yang berukuran 7 KB yang dijalankan dengan *interpreter* Python yang berukuran 26 KB.

## VI. Kesimpulan

Algoritma *link-state* telah diadaptasikan untuk aplikasi *proxy selector*, dengan menggunakan *moving average* untuk membantu pemilihan *proxy server* dengan pendekatan probabilistik. Berdasarkan perhitungan MSE dari data hasil eksperimen, panjang interval (*n*) *moving average* yang memberikan prediksi terbaik adalah 10. Prediksi yang terbaik dapat diberikan oleh panjang interval yang berbeda jika perhitungan dilakukan dengan menggunakan kumpulan data lain.

Pada sebagian besar kasus dalam eksperimen yang dilakukan, nilai *n* sebesar 10 paling banyak memberikan prediksi yang terbaik menurut perhitungan MSE. Nilai tersebut adalah nilai tertinggi dalam *range* nilai *n* yang diuji (2 sampai dengan 10). Karena itu, besar kemungkinan terdapat panjang interval *moving average* yang memberikan hasil prediksi lebih baik dari $n = 10$ untuk eksperimen ini jika *range* nilai *n* yang diuji diperbesar sampai lebih dari 10.

## Referensi


[1] A. Shaikh, D. Wang, G. Li, J. Yates, and C. Kalmanek, *An Efficient Algorithm for OSPF Subnet Aggregation*, 1998.

[2] Nortel Networks., *NetKnowledge: Routing*, Nortel Networks, 2002.

[3] N. E. Hwa, *Different Uses of Moving Average (MA)*, 2007.

[4] V. K. Boken, *Forecasting Spring Wheat Yield Using Time Series Analysis: A Case Study of Canadian Prairie*, Agronomy Journal, 2000.

[5] N. Bartolini, G. Bongiovanni, and S. Silvestri, *Distributed Server Selection and Admission Control in Replicated Web Systems*, IEEE Computer Society, 2007.

[6] L. D'Acunto, N. Chiluka, T. Vinkó, and H. Sips, *BitTorrent-like P2P approaches for VoD: A comparative study, Elsevier, 2013.*