MODIFIKASI *CACHE* DALAM PROSES PERTUKARAN DATA TERDISTRIBUSI DENGAN MENGGUNAKAN ALGORITMA KEMIRIPAN DOKUMEN

Rizal Fathoni Aji dan Zainal A. Hasibuan
{rizal,zhasibua}@cs.ui.ac.id
Fakultas Ilmu Komputer Universitas Indonesia
Kampus UI Depok 16424

ABSTRAKSI

Pada proses pertukaran data terdistribusi, data yang dialirkan ke dalam jaringan sangat besar. Salah satu cara yang dapat dilakukan untuk mengatasi masalah tersebut adalah menggunakan cache dalam proses pertukaran data. Namun, dalam proses pencarian dokumen terdistribusi, sangat jarang ditemukan query yang sama antara satu dengan yang lainnya, sehinga berakibat pada seringnya terjadi cache miss. Dalam makalah ini, cache akan divariasikan dengan cara menggunakan algoritma kemiripan dokumen untuk meminimalkan terjadinya cache miss. Hasil dari penggunaan algoritma tersebut akan dibandingkan efisiensinya dan dilihat pengaruhnya terhadap efektifitas hasil pencarian.

1. PENDAHULUAN

Dengan makin berkembangnya pertukaran data dan informasi dalam jaringan internet, dibuatlah sebuah cara agar terjadi efisiensi dalam penggunaan *bandwidth*. Efisiensi yang dapat dilakukan antara lain dengan menggunakan *cache*. *Cache* dikenal pada sistem operasi sebagai cara untuk menampung informasi yang sudah didapatkan sebelumnya. Namun, penggunaan *cache* sekarang makin meluas, terutama pada proses komunikasi [3,7,8,9].

Dalam komunikasi data, *cache* yang paling sering digunakan adalah *web cache*. Dari tesisnya, [7] menyebutkan bahwa pemakaian *web cache* menjadi poluler karena kemampuannya dalam menghemat *bandwidth*. Penghematan *bandwidth* salah satunya dipengaruhi oleh kemampuan *cache* dalam melakukan penggantian dokumen yang disimpannya. Makin efisien suatu *cache* dalam proses penggantian dokumen, makin kecil *bandwidth* yang digunakan.

2. *CACHE*

Ada beberapa algoritma penggantian yang dapat digunakan, yaitu *Least Recently Used* (LRU), *Least Frequently Used* (LFU), *First In First Out* (FIFO) dan *Largest Size Used* (LSU) (Suadi 2000). Masing-masing algoritma tersebut akan dijelaskan sebagai berikut.

- *Least Recently Used*

Inti dari algoritma ini adalah membuang dokumen yang sudah lama tidak digunakan. Idenya adalah, dokumen yang sudah lama tidak digunakan, akan lebih kecil kemungkinannya untuk diambil kembali, sehingga tempatnya dapat digantikan oleh dokumen baru yang kemungkinan akan diperlukan kemudian. Algoritma ini dianggap mendekati agoritma yang optimal [6].

- *Least Frequently Used*

Algoritma ini akan mengganti dokumen yang mempunyai jumlah frekuensi penggunaan yang kecil. Jumlah dokumen yang aktif digunakan dianggap memiliki peluang paling besar untuk digunakan kembali [7,9]. Menurut [7] algoritma ini memiliki kelemahan, karena dokumen yang pada awalnya aktif, tetapi lama tidak digunakan akan tetap dalam *cache* walau sudah jarang penggunaannya.

- *First In First Out*

Dasar dari algoritma ini adalah penggantian dokumen yang sudah lama berada dalam *cache*. Dokumen yang sudah awal dianggap tidak akan diambil lagi, sehingga digantikan oleh dokumen yang lebih baru [7,9]. Kelemahannya adalah, algoritma ini akan membuang dokumen jika waktunya telah habis, walaupun ia masih aktif digunakan [7].

- *Largest Size Used*

Dalam algortima ini, dokumen yang memiliki ukuran paling besar akan dibuang terlebih dahulu. Idenya adalah memasukkan sebanyak mungkin dokumen ke dalam *cache* sehingga kemungkinan *hit* akan lebih sering [7,9]. Menurut [7], kelemahan algoritma ini jika ada dokumen yang populer namun ukurannya besar, maka dokumen itu akan terus tergantikan dalam *cache*.

Cache memiliki mekanisme untuk menentukan apakah suatu pencarian sudah pernah dilakukan atau belum. Cara yang umum digunakan adalah dengan mencocokkan apakah query yang sama sudah pernah dilakukan sebelumnya atau belum. Jika sudah pernah dilakukan, maka terjadi *cache hit*, lalu hasil penelusuran di dalam *cache* akan dikembalikan ke pengguna. Sementara, jika query belum pernah dilakukan, akan terjadi *cache miss*, dan *system* akan mencari dokumen- dokumen hasil pencarian ke system.

3. MODIFIKASI *CACHE*

Pada umumnya pengguna menggunakan query yang berbeda-beda dalam melakukan pencarian dokumen, padahal query tersebut mirip antara satu dengan yang lainnya. Contohnya antara query 'penggunaan teknologi jaringan syaraf tiruan' dengan query 'teknologi jaringan syaraf tiruan dengan komputer', kedua query tersebut berbeda, tetapi memiliki kemiripan. Jika menggunakan algoritma penggantian *cache* yang umum, maka *cache miss* akan sering terjadi. Karena itu, *cache* perlu diubah agar algoritma penggantian *cache* lebih efisien.

Dalam mengukur kesamaan antara dokumen, yang umum digunakan adalah penggunaan *vector space model*. Pada *vector space model*, query-query direpresentasikan sebagai sistem koordinat, dimana kata-kata adalah sumbu dan query adalah vektor pada system koordinat tersebut. Untuk menghitung kesamaan antar query, digunakanlah cara yang paling mudah, yaitu dengan cara menghitung jarak antar vector-vektor query tersebut.

Cara yang mudah dalam mengukur jarak antar vector adalah menggunakan perhitungan *euclidean distance* [1] *Euclidean distance* dihitung dengan rumusan sebagai berikut:

$$distance(Q,D) = \sqrt{\sum_{k-1}^{n} (d_k - q_k)^2}$$

Dalam implementasinya, untuk menghitung kedekatan antar *query* dilakukan dengan perhitungan *euclidean distance*, dimana *d*k adalah jumlah *term* k di dalam query 1, dan *q*k adalah jumlah *term* k di dalam query 2.

4. EKSPERIMEN

Pada eksperimen ini akan dilihat pengaruh cache yang digunakan terhadap efisiensi dan efektifitas pertukaran data. Eksperimen ini dilakukan dengan cara mengirim *query* yang telah ditentukan sebelumnya ke sistem penelusur dokumen, selanjutnya sistem akan mengirim *query* ke masing-masing perpustakaan yang terhubung kepadanya. *Query* yang digunakan berupa nama-nama kelas yang diambil dari sub kelas puluhan pada klasifikasi DDC berbahasa inggris.

Setiap ujicoba, akan menggunakan parameter *treshold* yang berbeda untuk setiap algoritma. *Treshold* yang digunakan pada ujicoba ini merupakan perhitungan antara dua vektor, yaitu vektor *query* yang diberikan pengguna dengan vektor-vektor *query* yang sudah berada dalam *cache*. Dengan kata lain, *treshold* dengan mencerminkan kedekatan antara satu *query* dengan *query* yang lain. Makin kecil nilai *treshold*, makin dekat kemiripan antara kedua *query* tersebut.

Dalam *cache* ini, *treshold* digunakan dalam menentukan apakah *query* yang diterima memiliki kemiripan dengan *query* yang sudah berada dalam *cache*. Jika nilai kedekatan antar *query* tersebut masih berada dalam batas yang ditentukan, maka *cache* tidak perlu melakukan *request* ke masing-masing server, melainkan hanya mengembalikan data yang telah berada dalam *cache*. Sehingga, besar *treshold* akan mempengaruhi efisiensi dari *cache*.

Untuk melihat efisiensi penggunaan *cache* pada sistem, dalam ujicoba ini akan dihitung total *request* yang dikirim sistem ke setiap sumber data. Banyaknya *request* yang dikirim akan berbanding lurus dengan *bandwidth* yang digunakan. Sehingga, makin sedikit *request* yang dikirim makin kecil pula bandwidth yang digunakan dalam proses pengumpulan data.

Selanjutnya, dalam eksperimen ini juga, akan dilihat pengaruh efisiensi yang dilakukan terhadap efektifitas pencarian. Pengukuran efektifitas dilakukan dengan menghitung *precision* dan *recall* dari hasil pencarian. *Precision* dan *recall* dihitung dengan rumusan sebagai berikut:

Recall = A/B
Precision = A/C
Dimana:
A = jumlah dokumen relevan yang terambil oleh sistem
B = jumlah dokumen relevan yang terdapat pada koleksi dokumen
C = jumlah dokumen yang terambil

Relevansi *query* dengan dokumen yang terambil dicocokkan dengan cara menyamakan nomor kelas DDC dari query terhadap nomor kelas dari dokumen. DDC sudah umum digunakan dalam menentukan perbandingan antara satu dokumen dengan dokumen yang lain, seperti pada penelitian yang dilakukan oleh [4,5].

5. HASIL EKSPERIMEN

Dari ujicoba yang telah dilakukan, didapatkan hasil pada tabel 1 berikut.

Tabel 1. Hasil Eksperimen

	Total waktu (ms)	Waktu per *request* (ms)	Total *request*	Efisiensi (%)
Baseline	931100	3135.02	297	-
Euclidean 1	732488	2466.29	297	0.00
Euclidean 1.5	509379	2927.47	174	41.41
Euclidean 2	353917	3277.01	108	63.64
Euclidean 2.5	46559	3879.92	12	95.96

Waktu rata-rata per *request* dihitung dengan cara membagi waktu total dengan jumlah *request* yang dilakukan pada setiap ujicoba. Sedangkan perhitungan nilai efisiensi dilakukan dengan cara membandingkan jumlah *request query* pada baseline dengan jumlah *request query* pada ujicoba Perhitungannya dilakukan dengan rumusan sebagai berikut:

Efisiensi Ujicoba ke i = $\frac{(Rb - Ri)}{Rb} \times 100\%$

Dimana Rb adalah jumlah *request baseline* dan Ri adalah jumlah *request* pada ujicoba ke-i.

Pada tabel hasil uji coba diatas terlihat, makin besar *treshold* yang yang digunakan, makin sedikit *request* yang perlu dikirim. Jumlah *request* yang dilakukan akan berpengaruh kepada *bandwidth* dan waktu. Makin kecil *request* akan memperpendek waktu pencarian dan memperkecil *bandwidth* yang dibutuhkan.

Pada umumnya, penambahan efisiensi akan mengurangi efektifitas. Pada tabel selanjutnya, akan diperlihatkan perbandingan efektifitas pencarian yang dihitung dengan perhitungan *precision* dan *recall*. Hasil perhitungan dari setiap *query* di interpolasi untuk menghasilkan nilai yang sesuai untuk 11 titik *recall*. Selanjutnya nilai pada setiap titik *recall* dari semua *query* dirata-ratakan sesuai perhitungan pada [2] untuk menghasilkan tabel 2 berikut.

Tabel 2. Efisiensi dan efektifitas eksperimen

Algoritma	Rata-rata penurunan efektifitas (%)	Kenaikan Efesiensi (%)
Euclidean 1	2.62	0.00
Euclidean 1.5	31.72	41.41
Euclidean 2	44.20	63.64
Euclidean 2.5	49.92	95.96

Dari hasil ujicoba terlihat, kenaikan efisiensi akibat dari penggunaan *cache* akan menurunkan efektifitas hasil pencarian.

6. KESIMPULAN DAN SARAN

Dengan kondisi jaringan komputer di Indonesia yang masih serba kekurangan, proses pencarian sumber-sumber informasi tersebut akan membutuhkan kapasitas *bandwidth* yang besar. Karena itu diperlukan cara untuk dapat mengefisiensikan penggunaan *bandwidth* tersebut. Dari hasil eksperimen ini, dapat terlihat bahwa secara umum, peningkatan efisiensi akan menurunkan efektifitas hasil pencarian. Ini sesuai dengan *common sense* yang sudah ada sebelumnya, jika efisiensi makin besar, pasti akan mengorbankan efektifitasnya.

Untuk pengembangan selanjutnya, yang dapat dilakukan adalah pencarian algoritma-algoritma lain untuk meningkatkan efisiensi tanpa mengalami banyak penurunan pada efektifitas. Peningkatan efisiensi sangat diperlukan karena kondisi jaringan komputer dan internet di Indonesia masih kurang memadai. Selain itu, untuk pengembangan lebih lanjut, dapat juga dikembangkan *plugin* tambahan untuk protokol dan jenis metadata lainnya.

PUSTAKA

[1] Anton, Howard. 2000. *Elementary Linear Algebra*. London: Wiley and Sons

[2] Baeza-Yates dan Ribeiro-Neto, Ricardo dan Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. New York: Addison-Wesley Publishing Company

[3] Glassman, S. 1994. *A Caching Relay for the World Wide Web. Computer Networks and ISDN systems*. Proceeding of First International Conference on the World-Wide Web, Elsevier Science BV

[4] Jenkins, C., M. Jackson, P. Burden dan J. Wallis. 2006. *Automatic classification of Web resources using Java and Dewey Decimal Classification*. http://www7.scu.edu.au/1846/com1846.htm

[5] Lasic-Lazic, J., S. Seljan, H. Stancic. 2000. *Information Retrieval Techniques*. Croatia: Proceeding of 2nd CARNet Users Conference

[6] Silberschatz, Abraham. 1995. *Operating System Concept*. New York: Addison-Wesley Publishing Company

[7] Suadi, W. 2000. *Analisa Kinerja Algoritme Pengganti Halaman Pada Proxy Cache Server di Universitas Indonesia*. Tesis Pascasarjana Program Studi Ilmu Komputer. Fakultas Ilmu Komputer Universitas Indonesia

[8] Williams, S. 1995. *Caching Proxies: Limitations and Potentials*. Proceeding of 4th Int. World Wide Web Conference

[9] Williams, S., M. Abrams, C. R. Standridge, G. Abdulla, dan E.A. Fox. 1996. *Removal policies in network caches for World-Wide Web documents*. Proceeding of ACM SIGCOMM'96 Conference