Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik202182676
Vol. 8, No. 1, Februari 2021, hlm. 69-76 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 36/E/KPT/2019 e-ISSN: 2528-6579

PENINGKATAN KINERJA JARINGAN DENGAN MENGGUNAKAN MULTI-RULE ALGORITHM

Tanwir[1], Parma Hadi Rantelinggi*[2], Sri Widiastuti[3]

[1,3] Program Studi Teknik Elektro, Universitas Sains dan Teknologi Jayapura, Jayapura
[2] Program Studi Teknik Informatika, Universitas Papua, Manokwari
Email: [1]tanwir@ieee.org, [2]p.rantelinggi@unipa.ac.id, [3]sriwidiastuti@ustj-papua.ac.id
*Penulis Korespondensi

(Naskah masuk: 06 November 2019, diterima untuk diterbitkan: 01 Feburari 2021)

Abstrak

Algoritma pergantian adalah suatu mekanisme pergantian objek dalam cache yang lama dengan objek baru, dengan mekanisme melakukan penghapusan objek sehingga mengurangi penggunaan bandwidth dan server load. Penghapusan dilakukan apabila cache penuh sehingga penyimpanan entri baru diperlukan. Secara umum algoritma FIFO, LRU dan LFU sering digunakan dalam pergantian objek, akan tetapi diperoleh suatu objek yang sering digunakan namun terhapus dalam pergantian cache sedangkan objek tersebut masih digunakan, akibatnya pada waktu klien melakukan permintaan dibutuhkan waktu yang lama dalam browsing objek. Untuk mengatasi masalah tersebut dilakukan kombinasi algoritma pergantian cache Multi-Rule Algorithm, dalam bentuk algoritma kombinasi ganda FIFO-LRU dan triple FIFO-LRU-LFU. Algoritma Mural (Multi-Rule Algorithm) menghasilkan respon pada cache size 200 MB dengan waktu tanggapan rata-rata berturut-turut 56,33 dan 42 ms, sedangkan pada algoritma tunggal memerlukan waktu tanggapan rata-rata 77 ms. Sehingga Multi-Rule Algorithm dapat meningkatkan kinerja terhadap waktu penundaan, throughput, dan hit rate. Dengan demikian, algoritma pergantian cache Mural, sangat direkomendasikan untuk meningkatkan akses klien.

Kata kunci: *Algoritma Mural, Jaringan, Performa, Telekomunikasi, Throughput*

NETWORK PERFORMANCE IMPROVEMENT USING MULTI-RULE ALGORITHM

Abstract

Substitution algorithm is a mechanism to replace objects in the old cache with new objects, with a mechanism to delete objects so that it reduces bandwidth usage and server load. Deletion is done when the cache is full so saving new entries is needed. In general, FIFO, LRU and LFU algorithms are often used in object changes, but an object that is often used but is deleted in the cache changes while the object is still being used, consequently when the client makes a request it takes a long time to browse the object. To overcome this problem a combination of Multi-Rule Algorithm cache replacement algorithms is performed, in the form of a double combination algorithm FIFO-LRU and triple FIFO-LRU-LFU. The Mural algorithm (Multi-Rule Algorithm) produces a response on a cache size of 200 MB with an average response time of 56.33 and 42 ms respectively, whereas a single algorithm requires an average response time of 77 ms. So the Multi-Rule Algorithm can improve the performance of the delay, throughput, and hit rate. Thus, the Mural cache change algorithm, is highly recommended to improve client access.
Keywords: *Mural Algorithm, Network, Performance, Telecomucation, Throughput*

1. PENDAHULUAN

Perkembangan telekomunikasi seiring dengan perkembangan teknologi jaringan informasi (Rantelinggi, Paiki & Rantelobo, 2017), khususnya *internet*. Salah satu elemen penting dalam akses *Internet* adalah *Proxy Server* yang merupakan *server* yang dikonfigurasikan pada beberapa fungsi, termasuk sebagai *cache server* dan pengatur *bandwidth* (Tanwir, Hendrantoro & Affandi, 2015; Tanwir,dkk., 2017).*Cache* dalam hal ini merupakan tempat penyimpanan objek secara sementara untuk mempercepat perpindahan objek pada *Proxy Server*. Dengan adanya *cache*, setiap objek permintaan tidak secara langsung dilayani dengan mengakses *Internet*, tetapi didahului dengan mencoba mengakses proxy

cache lebih dahulu. Objek yang diminta ditemukan, menyebabkan waktu tanggapan terhadap permintaan akan menjadi lebih cepat dan penggunaan bandwidth saluran dapat dihemat. Karena ukuran *cache* yang terbatas, menyebabkan pengendalian penggunaan *bandwidth* dilakukan melalui penghapusan objek dengan menggunakan algoritma pergantian *cache*. Penghapusan dilakukan apabila *cache* sudah penuh dan diperlukan penyimpanan entri baru.

Besarnya ukuran dalam proses penyimpanan objek pada *cache proxy* (Baek, dkk 2015) dilakukan dengan menyimpan objek-objek yang pernah diakses (Chen,dkk., 2016). Apabila layanan ke *internet* klien berisi permintaan atas objek yang telah diminta lebih awal dan sudah tersimpan di *cache*, sehingg *Proxy Server* bisa langsung menyerahkan objek dari *Cache* yang diminta pada klien ke *server* tanpa harus *server* asalnya minta ulang lewat *internet*. Unjuk kerja tersebut dimana objek yang diperlukan tidak ada dalam *cache* pada *Proxy Server*, selanjutnya *Proxy Server* melakukan permintaan ke *server* asal di *internet* (Xia, dkk., 2016). *Proxy Server* disini berfungsi sebagai penyimpanan *cache* untuk objek yang memiliki kemungkinan permintaan oleh komputer klien ke *internet* publik. Dengan mengalihkan http, *proxy* selalu memberikan objek terbaru, karena *Proxy Server* selalu menyesuaikan objek yang ada dalam *cache* dengan objek yang berada pada *server* luar. Sedangkan efektifitas dan validasi algoritma pergantian *cache* digunakan untuk menganalisa dan membandingkan kinerja *Proxy Server* tersebut berdasarkan permintaan hit dan waktu tanggapan (Yen & Chien, 2018).

Tiga algoritma yang umum dipakai dalam proses pergantian *cache* adalah *first in first out* (FIFO), *least recently used* (LRU) dan *least frequently used* (LFU). Algoritma FIFO adalah algoritma yang paling simpel karena aturannya mirip aturan antrian tak diutamakan. Objek yang masuk lebih awal akan keluar lebih duluan. Algoritma ini menggunakan susunan data stack apabila tidak ditemukan bingkai yang kosong ketika terjadi page fault, yang dipilih adalah bingkai dengan stack terbawah, seperti halnya objek yang telah lama disimpan dalam *cache*. Karena hal tersebut algoritma ini dapat menempatkan ke tempat lain page yang sering digunakan. Kelemahan algoritma FIFO adalah performanya yang kadang tidak selalu bagus. Penyebabnya karena ada objek yang belum lama keluar dari *cache* ternyata dibutuhkan kembali (Shivaram, dkk., 2018).

Algoritma LRU memilih objek tidak digunakan pada kurun waktu yang lama. Keunggulan LRU adalah tidak akan mengalami anomali Belady, dan hanya membuang objek yang sudah lama tidak diakses, tanpa memandang kapan objek tersebut masuk pada *cache* disk pertama kali. Implementasi LRU dilakukan dengan penggunaaan stack yang menandakan objek-objek berada di *cache* disk. Setiap kali suatu objek diakses, kemudian diletakkan pada posisi paling atas pada stack. Apabila objek yang terletak paling bawah pada stack sudah melewati suatu batas waktu maksimum, sehingga objek tersebut akan diganti. Setiap kali terdapat objek baru yang akan diakses, segera menentukan objek sebagai pengganti karena sudah terdeteksi berdasarkan waktu kadaluarsa. Setiap kemunculan objek baru, algoritma ini melakukan dua kali loop: yang pertama adalah untuk mencari objek yang lama, diikuti yang kedua yaitu pergantian objek, sehingga memerlukan waktu yang lebih lama dalam pergantian objek dibandingkan FIFO.

Sedangkan algoritma LFU mengganti objek yang paling sedikit digunakan. LFU mendeteksi objek yang tidak digunakan dalam waktu yang ditentukan dengan fetch count ≤ 4. Selain itu algoritma LFU cukup sederhana dengan implementasi yang sangat efisien karena tidak membutuhkan banyak langkah dalam pemilihan objek. Sedangkan kelemahannya adalah bahwa objek yang dibuang karena penggunaannya dianggap sedikit mungkin sebenarnya masih aktif digunakan.

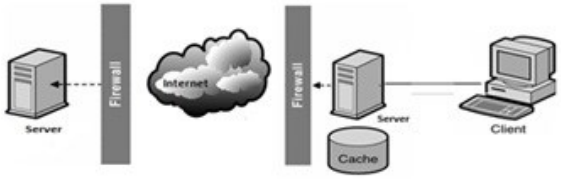
Dalam penelitian sebelumnya telah dicoba penggabungan FIFO-LRU untuk memanfaatkan keunggulan masing-masing. Dalam penelitian tersebut kami telah mengkombinasi dua algoritma pergantian *cache* dan melakukan perbandingan berdasarkan (Ajorloo & Manzuri-Shalmani, 2016; Ji, Caire & Molisch, 2015) parameter *throughput*, (Wang, dkk., 2017; Lee & Molisch, 2018) prosentase *hit rate*, waktu tanggapan dan waktu penundaan (Tanwir, Gamantyo & Affandi, 2017). Ternyata dengan penggabungan dua algoritma tersebut diperoleh adanya peningkatan efisiensi *bandwidth* dalam proses pergantian *cache* dibandingkan dengan algoritma tunggal, dimana *throughput* pada jaringan *server* meningkat terhadap ukuran *cache*. Selain itu terjadi penurunan pemakaian *bandwidth* yang menyebabkan waktu penundaan berkurang serta menghasilkan miss rate yang semakin kecil dan prosentase *hit rate* yang meningkat.

Studi literatur oleh (Tanwir, Gamantyo & Affandi, 2015, 2017) mengindikasikan bahwa kombinasi tiga algoritma LRU, LFU dan FIFO pada *cache server* dari suatu jaringan telekomunikasi menjanjikan peningkatan unjuk kerja *cache* dalam pergantian *cache* pada jaringan *internet* dan memberikan pengaruh yang besar pada pemakaian *bandwidth* dalam jaringan serta prosentase *hit rate* pada *cache*. Dari hasil awal selanjutnya penelitian dilakukan dengan mengusulkan metode kombinasi tiga algoritma tersebut dan mengevaluasi peningkatan unjuk kerja gabungan tiga algoritma tersebut yang dilaporkan dalam penelitian ini.

Oleh karena itu, kontribusi orisinil penelitian ini adalah metode penggabungan dua algoritma FIFO-LRU dan tiga algoritma FIFO-LRU-LFU yang disebut *Multi-Rule Algorithm* belum pernah diakses sebelumnya. Hasil evaluasi menunjukkan bahwa terjadi peningkatan unjuk kerja pergantian *cache* pada *Proxy Server* dibandingkan algoritma-algoritma tunggal maupun kombinasi dua algoritma. Sebagai akibatnya, terjadi peningkatan *throughput* dan efisiensi bandwith serta miss rate yang semakin kecil.

2. PROXY CACHE

Proxy cache memegang peranan penting dalam meningkatkan kinerja suatu komputer, *Proxy* merupakan aplikasi yang menjadi perantara antara klien dengan web *server*. Salah satu fungsi *proxy* adalah menyimpan cache yang telah digunakan. Pada hubungan klien dengan *server* pada gambar 1, klien meminta akses URL web sehingga *browser* mengirim permintaan ke *server*, apabila tidak ada, *server* langsung dapat meminta ke web *server* pada *cache* sehingga mempercepat proses *browsing*. Pada ilustrasi gambar *proxy* dimana *Proxy Server* merupakan dua keadaan yang berbeda. *Proxy* merupakan layanan yang dimiliki *Proxy Server*, sedangkan *Proxy Server* dapat melayani permintaan dari klien. Pada waktu dilakukan akses objek tertentu, internet protokol akan mengidentifikasi suatu objek, proses ini memberi informasi diijinkan menerima hasil permintaan atau tidak terhadap akses web. Seperti yang digambarkan pada Gambar 1.



Gambar 1. Hubungan Klien - *Server*

Gambaran *Proxy* sebagai media pihak ketiga yang terkait ditengah antara kedua pihak yang berhubungan, antara pihak pertama dan pihak kedua tidak terjadi mekanisme hubungan secara langsung, tetapi masing-masing terhubung dengan perantara, yaitu *Proxy*.

Dengan demikian *proxy* berada pada tingkat yang berbeda pada hirarki lapisan protokol komunikasi jaringan dan bekerja dalam berbagai jenis protokol komunikasi jaringan. *Proxy Server* sebagian besar ditujukan untuk menunjuk *server* yang menggunakan *proxy* dalam lapisan aplikasi.

Pada jaringan lokal yang terkoneksi ke jaringan *internet*, klien tidak dapat langsung terhubung dengan jaringan luar, tetapi melewati *gateway*, yang mengerjakan sebagai batas antara jaringan lokal dan jaringan luar. Peranan *Gateway* sebagai jaringan lokal harus dilindungi dengan baik dari bahaya yang mungkin berasal dari *internet*, dan hal tersebut sulit ditangani apa pun apabila tidak ada batas yang jelas antara jaringan lokal dan *internet*. *Gateway* juga memiliki fungsi sebagai titik dimana sejumlah koneksi dari klien lokal akan terhubung ke *Gateway*, dan begitu pula dengan jaringan luar juga terkoneksi ke *Gateway*. Apabila hubungan jaringan lokal ke *internet* memanfaatkan sambungan yang dipunyai oleh *gateway* secara bersama. Dalam hal ini, g*ateway* berfungsi sebagai *proxy server*, yang menyediakan layanan antar jaringan lokal dan jaringan luar.

Sebagai perantara *server* dan klien di *internet*, proses kerja *Proxy Server* dengan pola klien melakukan permintaan layanan, kemudian sebagai pengganti *Proxy Server* akan mewakili permintaan klien, ke *server* di *internet* yang maksud. Terkait *Proxy Server* sebenarnya hanya melanjutkan permintaan klien ke *server* yang tuju, tetapi disini identitas peminta sudah berubah, bukan lagi klien asal, akan tetapi *Proxy Server* tersebut. *Server* di *internet* hanya mengetahui identitas *Proxy Server* tersebut, sebagai yang melakukan permintaan, akan tetapi tidak tahu prrmintaan sebenarnya (yaitu klien asalnya) karena permintaan yang sampai ke *server* di *internet* bukan lagi dari klien asal, melainkan dari *Proxy Server*.

Pada proses yang terjadi pada *Proxy Server* pada pengguna sendiri tidak kelihatan (transparan), dimana klien melakukan permintaan atas layanan di *internet* langsung ke *server-server* layanan di *internet*. Penguna hanya dapat mengetahui alamat dari *Proxy Server*, yang diperlukan untuk mengatur pada sisi klien dalam menggunakan layanan dari *Proxy Server* tersebut.

Fungsi dasar yang sangat penting dari *Proxy Server* sebagai caching. *Proxy Server* memiliki cadangan untuk menyimpan objek-objek yang dimiliki oleh *server* di *internet*, secara teknis disebut caching. Oleh karena itu, *Proxy Server* juga melakukan proses caching yang disebut *cache server*.

Proses kerja caching akan menyimpan objek yang adalah hasil permintaan dari klien, yang diperoleh dari *internet*. Unjuk kerja *Proxy Server* sebagai media penengah, selanjutnya *Proxy Server* menerima objek tersebut dari sumber, kemudian melanjutkan ke peminta yang sebenarnya. Pada proses tersebut, *Proxy Server* juga menyimpan objek tersebut dalam ruang disk yang disediakan (*cache*).

Dengan demikian, pada waktu klien melakukan permintaan terhadap layanan ke *internet* yang berisi objek yang mirip dan pernah terjadi sebelumnya serta berada di dalam *cache*, *Proxy Server* dapat langsung menyediakan objek dari *cache* yang dikirimkan kepada klien, tanpa harus mulai ke *server* asal di *internet*. Apabila melakukan permintaan dan tidak ditemukan dalam *cache* di *Proxy Server*, baru kemudian *Proxy Server* pindah atau memintakannya ke *server* yang diterima di *internet*.

Proses caching ini juga tidak tampak klien (transparan), karena bagi klien tidak kelihatan siapa yang memberikan objek yang dimintanya, apakah *Proxy Server* yang mengambil dari *cache*-nya atau dari *server* asli di *internet*. Dari sisi klien, semua akan melihat langsung dari *internet*.

Saat klien dibuka dan mengenalkan URL pada *browser*, konten yang ditampilkan pada URL disebut "*Internet* Objek". Pertama dia akan bertanya terlebih dahulu ke *Domain Name Server* (DNS). Kemudian DNS akan mencari alamat *internet protocol* (IP) dari URL tersebut di dalam data basenya, setelah itu memberi jawaban kepada *browser* tersebut kembali. Kemudian *browser* mendapatkan alamat IP, selanjutnya klien akan membuka koneksi ke port http web *server* tujuan. Web *server* kemudian menyetujui permintaan dari *browser* lalu memberi konten yang mendukung tersebut. Hubungan web *server* bisa diputus menyetujui *browser*

yang telah menerima konten. Hardisk kemudian akan menyimpan dan menampilkan konten tersebut.

Konten yang disimpan dalam hardisk disebut *cache* objek yang nantinya akan digunakan saat klien kembali mendatangi objek yang sama, sebagai contoh aktivitas mengklik tombol kembali atau melihat history. Dalam kunjungan selanjutnya, *browser* akan melakukan pemeriksaan validasi konten yang dikirim, selanjutnya hasil validasi ini dilakukan dengan membandingkan header konten yang ada pada *cache* objek dengan yang ada pada web *server*, konten apa pun dianggap belum kadaluwarsa, sehingga konten tersebut akan diperlihatkan kembali ke *browser*.

Dalam hardisk lokal yang menyimpan *Cache* objek ini hanya biasa dipakai oleh klien secara pribadi, tidak dapat dibagi dengan klien yang lainnya, berbeda dengan konten yang tersimpan di *server,* apabila terkoneksi dengan semua komputer dan *server*, sehingga *cache* objek dapat digunakan bersama-sama, *server* ini yang dinamakan *cache server* atau *Proxy Server*.

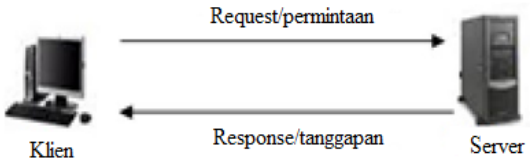
2.1 Throuhgput

Berbicara tentang *throughput* tentunya membahas lebih dahulu tentang suatu *bandwidth*, *bandwidth* adalah ukuran dari banyaknya data dalam bentuk bit yang dapat dikirim dari sumber ke tujuan dalam satu detik. *Bandwidth* pada dasarnya dimanfaatkan untuk mengukur aliran data analog dan data digital. Satuan yang digunakan dalam *Bandwidth* adalah *bit per second* (bps). Pengertian dari *Bandwidth Internet* adalah batas maksimal kecepatan yang diberikan oleh sebuah Penyedia jasa layanan *Internet*. Secara umum *bandwidth* dibedakan atas *bandwidth* Analog yang merupakan rentang antara frekuensi yang terendah dengan tertinggi yang dimanfaatkan pada transmisi informasi (digital ataupun analog) dengan satuan Hertz (Hz) yang berfungsi menentukan banyaknya informasi yang mampu ditransmisikan dalam suatu waktu. Selain itu ada *bandwidth* Digital yang merupakan banyaknya data (bit) yang bisa dikirimkan dan diterima lewat sebuah kanal komunikasi tanpa adanya distorsi.

Throughput sebuah sistem didefinisikan sebagai ukuran sebenarnya dari informasi yang dikirimkan melalui suatu saluran (Rantelinggi & Djanali, 2015). *Throughput* dapat diukur dalam satuan *bits/second* atau *packet/second*.

2.2 Waktu Tanggapan

Waktu tanggapan merupakan parameter yang digunakan untuk mengetahui berapa lama waktu yang diperlukan klien untuk melakukan permintaan objek, baik objek yang harus disimpan dalam *cache* juga yang belum masuk ke dalam *cache* sampai klien mendapatkan tanggapan atas permintaan seperti pada Gambar 2, dengan *hit ratio* sebesar persentase dari objek yang diambil dari *cache* dibandingkan dengan jumlah permintaan yang dikirim oleh *client* ke *cache*.



Gambar 2. Waktu Tanggapan

Dalam menggunakan objek yang berbeda-beda dalam melakukan pencarian objek, dimana objek tersebut mirip antara satu dengan yang lainnya. Apabila menggunakan algoritma penggantian *cache* yang umum, menyebabkan *cache* akan sering terjadi miss. Dimana *cache* miss merupakan objek yang diminta tidak berada dalam *cache*, harus diambil pada unit dibawahnya yang cukup memakan waktu. Ini disebut *miss* (gagal) sedangkan *cache* hit adalah suatu kondisi yang terjadi kompilasi dimana akses yang diminta berada dalam *cache* tersebut sehingga dengan cepat mengembalikan objek yang diminta. Karena itu, *cache* perlu diubah agar algoritma penggantian *cache* lebih efisien. Dalam pengukuran kesamaan antara objek, yang umum digunakan adalah penggunaan model jarak vektor. Pada model jarak vektor, objek direpresentasikan sebagai sistem koordinat, sedangkan objek-objek adalah sumbu yang merupakan vektor pada sistem koordinat tersebut. Untuk menghitung kesamaan antar objek, digunakan cara yang paling mudah, dengan cara menghitung jarak antar vector-vektor objek tersebut. Cara yang mudah dalam mengukur jarak antar vektor menggunakan perhitungan jarak euclidean dengan kedekatan antar objek dilakukan dengan jumlah objek.

2.3 Kode Pseudo

Sebuah kode yang pada umumnya untuk menulis sebuah algoritma dengan aturan yang bebas serta tidak terkait dengan bahasa pemrograman tertentu. Sekumpulan langkah untuk menyelesaikan suatu masalah, hanya polanya sedikit beda dari algoritma, kode Pseudo menyerupai bahasa pemrograman. Selain itu umumnya pseudo-code memanfaatkan bahasa yang sederhana dan lebih ringkas dari pada algoritma.

Kode pseudo merupakan penjelasan sederhana atas sebuah algoritma bahasa pemograman yang terstruktur, yang bertujuan agar dapat di mengerti oleh manusia. Kode pseudo pada umumnya tidak membutuhkan elemen detail agar manusia dapat paham suati algoritme seperti deklarasi variable, kode maupun subrutin yang spesifik. Bahasa yang digunakan di deskripsikan dalam bahasa natural atas sesuatu yang bersifat detail, serta menggunakan notasi matematik. Tujuannya untuk memudahkan pengguna agar paham dibanding menggunakan bahasa pemograman, karena ringkas dan tidak bergantu pada suatu algoritma. Kode Pseudo banyak digunakan dalam dalam publikasi ilmiah serta perencanaan pembangunan suatu program komputer sebelum kode program sesungguhnya di tulis.

Penggunaan kode pseudo pada buku teks dan publikasi ilmiah secara umum membahas kode pseudo

untuk menjelaskan suatu algoritma agar mudah dipahami oleh programmer. Dalam buku teks, kode pseudo banyak membahas tentang notasi dan konvensi, kadang digunakan pula suatu bahasa pemograman sebagai pendekatan. Dalam implementasi suatu algoritma yang spesifik, umumnya kode pseudo mengartikan algoritma tersebut agar dapat dimanfaatkan secara benar saat diterjemahkan ke dalam bahasa pemograman.

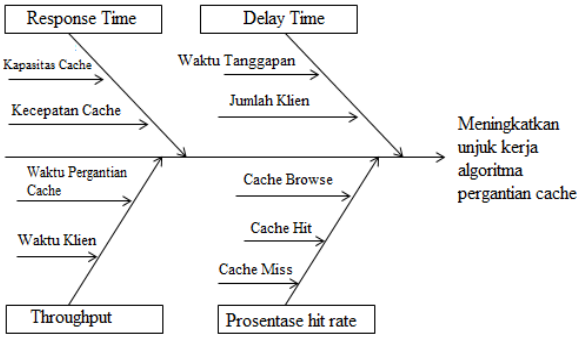
Programmer dapat mulai dari membuat sketsa kode di atas kertas sebelum diterjemahkan ke dalam bentuk kode bahasa pemograman, dengan pendekatan "*top-down*". Kode pseudo secara umum tidak mengikuti aturan yang berlaku dalam bahasa pemograman, dengan kata lain dalam bentu standar sistematik, meskipun penulisannya menggunakan aturan sintaksis, sebagai contoh struktur kontrol dari bahasa pemograman yang umum digunakan. Begitu pula halnya blok kode sering diganti dengan suatu baris penjelasan dalam bahasa yang dipahami atau natural.

Pola dari kode pseudo sangat bervariasi, dari model yang sederhana hingga yang mirip bahasa pemograman yang sesunggunya.

3. EVALUASI DAN DISKUSI

Bentuk aktivitas pengiriman objek pada *cache* mencari apakah objek tersebut ada di *cache*, apabila ada dikenal sebagai HIT dan tidak sebagai MISS, hal ini berupa *cache* kosong, kemudian *cache* dalam keadaan penuh dimana objek yang diinginkan seharusnya berada dalam cache tersebut sehingga cache harus mengganti salah satu objek dengan objek baru. Pada waktu terjadi ketika objek yang diinginkan tidak cukup untuk diisikan ke dalam cache, biasanya terjadi akibat objek yang cukup besar sehingga pemasukan objek ke dalam *cache* dilakukan dengan menggunakan alamat dengan *type field* yaitu *Tag, Line* dan *Word* yang terdapat pada lokasi *cache* sehingga objek dengan kapasitas besar akan menempati saluran *cache* dengan menginterpretasikan alamat sebagai *field Tag* dan *field Word* yang diidentifikasikan sebagai blok utama dan memastikan apakah suatu blok berada dalam *cache* serta setiap tag saluran yang sesuai telah diperiksa. Pemetaan ini dilakukan dalam saluran *cache* dalam bentuk pemetaan langsung (*direct mapping*), pemetaan asosiatif (*assosiative mapping*) dan pemetaan kelompok *(set associative mapping*). Pemetaan ini pada suatu blok *cache* penuh, mengizinkan menempatkan blok tersebut untuk dimuat ke sembarang saluran cache. Hal ini terdapat fleksibilitas pergantian blok baru dibaca dalam *cache* menyebabkan pencarian objek di *cache* menjadi lama. Selanjutnya ketiga pemetaan tersebut menentukan baris mana pada *cache* yang dapat diganti dengan blok *cache* baru. Apabila semua telah dilakukan akan terjadi algoritma pergantian cache LRU, LFU, FIFO, FIFO-LRU dan FIFO-LRU-LFU.

Suatu metode pergantian objek dalam cache yang lama dengan data baru disebut algoritma penggantian. Dalam gambaran ini, tidak membutuhkan algoritma ini, tetapi dalam pemetaan assosiatif dan asosiatif set, algoritma ini sangat membantu untuk meningkatkan kinerja cache. Telah banyak algoritma penggantian yang dikembangkan, tetapi algoritma yang paling efektif adalah *Least Recently Used* (LRU), yaitu algoritma dengan pola mengganti objek yang terlama dalam cache dan tidak memiliki referensi. Selain LRU ada algoritma lainnya yaitu *First In First Out* (FIFO), merupakan algoritma denga pola kerja mengganti objek yang masuk pertama. *Kemudian Least Frequently Used* (LFU) adalah algoritma dengan metode kerja mengubah blok data, tetapi algoritma LFU mempunyai referensi paling sedikit. Ketiga algoritma yang dimanfaatkan dengan menggunakan empat parameter antara lain *throughput*, prosentase *hit rate*, waktu penundaan dan waktu tanggapan yang digambarkan dalam bentuk Gambar 3 dengan parameter algoritma pergantian *cache*.

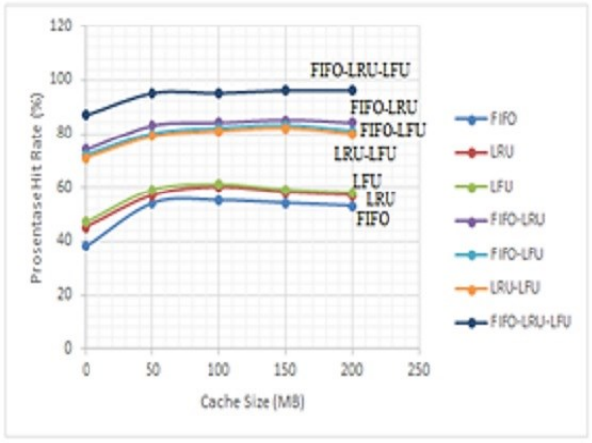


Gambar 3. Parameter Algoritma Pergantian Cache

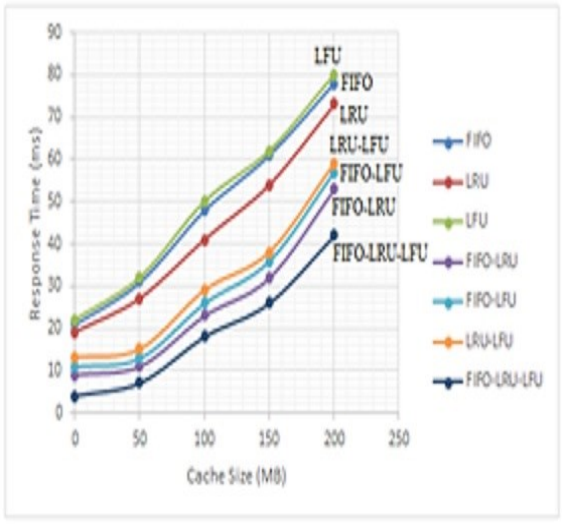
Dengan menggunakan perangkat *internet* yang dihubungkan pada 3 buah laptop dengan IP address 192.168.0.257, 192.168.0.258, 192.168.0.259 pada proses pengujian dengan spesifikasi laptop yang sama terhadap objek-objek yang melakukan permintaan dari tiga buah laptop tersebut, masing-masing dengan objek yang sama dengan variabel ukuran cache 50 MB, 100 MB, 150 MB dan 200 MB diperoleh algoritma pergantian cache FIFO-LRU-LFU menghasilkan respon yang baik dan cepat sehingga prosentase *hit rate* terhadap ukuran cache semakin meningkat. Dari pengamatan dihasilkan bentuk grafik prosentase *hit rate*, waktu penundaan dan *throughput* terhadap ukuran *cache* seperti pada Gambar 4, 5 dan 6.

Gambar 4 menunjukkan bahwa prosentase *hit rate* meningkat terhadap ukuran *cache* sampai pada ukuran *cache* sekitar 100 MB, setelah itu prosentase *hit rate* memiliki nilai hampir konstan. Hal ini dapat dijelaskan sebagai berikut. Peningkatan jumlah permintaan yang dapat dilayani *cache* ("hit") menambahkan jumlah prosentase objek yang diambil dari *cache* terhadap jumlah total permintaan yang dikirim oleh klien ke *cache*. Prosentase *hit rate* tertinggi ditunjukkan oleh *Multi-Rule Algorithm Algorithm* (Kombinasi FIFO-LRU-LFU), yang mengindikasikan peningkatan unjuk kerja karena pada algoritma ini sejumlah objek tidak langsung terhapus dan meningkatkan peluang hit. Untuk ukuran *cache* sebesar 100 – 250 MB, prosentase *hit rate*

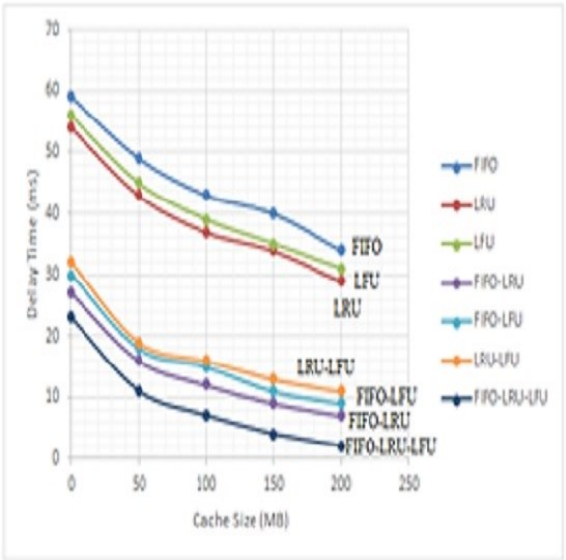
dengan *Multi-Rule Algorithm* (Kombinasi FIFO-LRU-LFU), mampu mencapai 96%.



Gambar 4. Prosentase *Hit Rate Multi-Rule Algorithm*



Gambar 5. *Response Time* terhadap *Cache Size*



Gambar 6. Waktu Penundaan *Multi-Rule Algorithm*

Sedangkan berdasarkan hasil pengukuran permintaan awal objek dengan ukuran *cache* 50 MB terhadap *Multi-Rule Algorithm* (Kombinasi FIFO-LRU-LFU) dengan sejumlah permintaan yang telah dilakukan menghasilkan prosentase perbandingan *cache* hit terhadap banyaknya permintaan sebesar 38% dengan algoritma FIFO, 45% dengan LRU, 47% dengan LFU. Sedangkan implimentasi kombinasi algoritma FIFO-LRU menghasilkan prosentase *hit rate* sebesar 74% dan prosentase hit tersebut semakin meningkat mencapai 87% pada *Multi-Rule Algorithm* (Kombinasi FIFO-LRU-LFU).

Seperti yang ditunjukkan oleh Gambar 5 waktu tanggapan meningkat terhadap ukuran *cache*. Gejala ini disebabkan karena dengan ukuran *cache* yang makin besar diperlukan waktu yang lebih panjang pula untuk melakukan pencarian objek. Dari hasil analisis data awal ukuran *cache* 50 MB diperoleh waktu tanggapan FIFO 21 ms, LRU 19 ms, LFU 22 ms sedangkan dengan implementasi kombinasi dua algoritma FIFO-LRU dihasilkan waktu tanggapan sebesar 9 ms, FIFO-LFU 11 ms, LRU-LFU 13 ms. Dengan kombinasi tiga algoritma FIFO-LRU-LFU diperoleh waktu tanggapan yang hanya sebesar 4 ms, jauh di bawah nilai waktu tanggapan untuk algoritma-algoritma yang lain.

Gambar 6 menunjukkan bahwa dengan ukuran *cache* 50 MB terjadi waktu penundaan dengan algoritma FIFO 59 ms, LRU 53 ms, LFU 54 ms dan FIFO-LRU 26 ms, FIFO-LFU 30 ms, LRU-LFU 32 ms dengan kombinasi tiga algoritma FIFO-LRU-LFU 23 ms Pada ukuran *cache* 150 MB diperoleh waktu penundaan FIFO 43 ms, LRU 37 ms, LFU 37 ms sedangkan kombinasi FIFO-LRU mengalami penurunan menjadi 12 ms, FIFO-LFU 15 ms, LRU-LFU 16 ms dan FIFO-LRU-LFU menjadi 7 ms. Dengan demikian terdapat fenomena bahwa ukuran *cache* yang semakin besar akan memperkecil miss rate, mengakibatkan waktu penundaan mengalami penurunan terhadap ukuran *cache*, sehingga waktu pencarian objek menjadi lebih cepat.

Waktu penundaan mengalami penurunan ketika diterapkan kombinasi algoritma FIFO-LRU-LFU, seperti yang diindikasikan dalam Gambar 6. Dengan ukuran *cache* yang makin besar, untuk melakukan permintaan dibutuhkan waktu semakin besar yang sebanding dengan ukuran *cache*. Setelah permintaan terhadap objek tersebut mengalami hit, objek akan tersimpan dalam *cache* disk yang menyebabkan penurunan waktu ketika dilakukan permintaan kembali terhadap objek yang sama. Ketika penyimpanan *cache* telah penuh, dilakukan penghapusan objek dengan implementasi kombinasi algoritma FIFO-LRU-LFU berdasarkan *clock* logika *counter* dan *stack* terhadap antrian objek sehingga menghasilkan penurunan waktu waktu penundaan yang lebih jauh terhadap ukuran *cache*.

Gambar 6 menunjukkan bahwa *Multi-Rule Algorithm* (Kombinasi FIFO-LRU-LFU) lebih unggul dibandingkan algoritma-algoritma lain dalam hal *throughput*. Dengan ukuran *cache* 50 MB dicapai *throughput* dengan algoritma-algoritma tunggal sebesar 20-23 kbps. Dengan kombinasi dua algoritma terjadi

peningkatan *throughput* sampai antara 40-45 kbps, sedangkan dengan *Multi-Rule Algorithm* (Kombinasi FIFO-LRU-LFU) dicapai *throughput* sebesar 66 kbps *Throughput* mengalami peningkatan pada ukuran *cache* 150 MB menjadi 28-32 kbps dengan algoritma-algoritma tunggal, sedangkan dengan kombinasi dua algoritma *throughput* dapat mencapai 50-56 kbps. *Throughput* semakin meningkat dengan Algoritma Mural (Kombinasi FIFO-LRU-LFU). mencapai 85 kbps karena peluang yang semakin besar untuk menemukan objek yang dipermintaan di dalam *cache* akan mengurangi waktu akses ke *server* dan meningkatkan *throughput*.

Sedangkan perbandingan Prosentase *Hit Rate* pada *cache* size 200 MB diperoleh rata-rata 56 % pada algoritma tunggal, 81,25 % pada algoritma ganda dan 96 % pada algoritma *triple*. Perbandingan prosentase *hit rate* algoritma tunggal dengan algoritma ganda sebesar 25,25 %, sedangkan perbandingan algoritma tunggal dengan *triple* meningkat menjadi 40 % dan perbandingan algoritma ganda dengan *triple* sebesar 14,75 %. Sehingga prosentase *hit rate* mengalami peningkatan dari pergantian *cache* pada algoritma tunggal terhadap kombinasi pergantian *cache* sebesar 54,75 %.

Rata-rata waktu penundaan pada *cache* size 200 MB pada algoritma tunggal 31,33 ms, pada algoritma ganda 9 ms dan algoritma *triple* sebesar 2 ms, diperoleh perbandingan waktu penundaan algoritma tunggal terhadap algoritma ganda sebesar 22,33 ms, perbandingan algoritma tunggal terhadap algoritma *triple* sebesar 29,33 ms dan perbandingan algoritma ganda dengan *triple* diperoleh 7 ms. Sehingga waktu penundaan semakin mengalami penurunan dari algoritma tunggal menjadi kombinasi algoritma sebesar 2 ms.

4. KESIMPULAN

Perbandingan Prosentase *Hit Rate* pada *cache* size 200 MB prosentase *hit rate* mengalami peningkatan sebesar 54,75 % terhadap pergantian algoritma yang digunakan, dimana waktu penunndaan pada *cache* size yang sama waktu penundaan pergantian algoritma sebesar 2 ms, diperoleh peningkatan *throughput* sebesar 44,33 Kbps, dengan waktu tanggapan sebesar 42 ms, menyebabkan meningkatnya prosentase *hit rate* dengan demikian terjadi peningkatan kecepatan *request* dengan waktu respon semakin kecil. Akibatnya waktu penundaan mengalami penurunan terhadap *cache* size sehingga waktu pencarian objek menjadi lebih cepat.

Dengan pengembangan kombinasi algoritma pergantian *cache* FIFO-LRU-LFU dalam bentuk *Multi-Rule Algorithm* meningkatnya prosentase *hit rate* sehingga waktu yang dibutuhkan dalam pencarian objek semakin kecil. Hasilnya diperoleh penggunaan algoritma gabungan FIFO-LRU-LFU jauh lebih baik apabila dibandingkan algoritma gabungan FIFO-LRU dan algoritma FIFO, LRU dan LFU sangat direkomendasikan untuk meninkatkan *throughput* akses pada klien.

DAFTAR PUSTAKA

AJORLOO, H. & MANZURI-SHALMANI, M.T., 2016. Throughput Modeling of Distributed Reservation Protocol. IEEE Transactions on Mobile Computing, 15(2), pp.503–515.

BAEK, LEE, NICOPOULOS, LEE, KIM., 2015. Size-Aware Cache Management for Compressed Cache Architectures. IEEE Transactions on Computers, 64(8), pp.2337-2352

CHEN, XIAO, LU, LIU., 2016. Me-CLOCK:A Memory-Efficient Framework to Implement Replacement Policies for Large Caches. Conference Name: IEEE Transactions on Computers, 65(8), pp. 2665-2671.

CAO, ZHANG, CHEN, LIU, KANG, GÜNDÜZ., 2019. Coded Caching With Asymmetric Cache Sizes and Link Qualities: The Two-User Case. IEEE Transactions on Communications, 67(9), pp.6112-6126

JI, M., CAIRE, G. & MOLISCH, A.F., 2015. The Throughput-Outage Tradeoff of Wireless One-Hop Caching Networks. IEEE Transactions on Information Theory, 61(12), pp.6833–6859.

LEE, MOLISCH., 2018. Caching Policy and Cooperation Distance Design for Base Station-Assisted Wireless D2D Caching Networks: Throughput and Energy Efficiency Optimization and Tradeoff. IEEE Transactions on Wireless Communications, 17(11), pp.7500-7514

NIKOLAOU, VAN RENESSE, SCHIPER., 2016. Proactive Cache Placement on Cooperative Client Caches for Online Social Networks. IEEE Transactions on Parallel and Distributed Systems, 27(4), pp.1174-1186

RANTELINGGI, P.H. & DJANALI, S., 2015. Kinerja Protokol Routing Pada Lingkungan Wireless Mesh Network Dengan Combined Scalable Video Coding. JUTI: Jurnal Ilmiah Teknologi Informasi, 13(1), pp.86-94–94.

RANTELINGGI, P.H., PAIKI, F.F. & RANTELOBO, K., 2017. Performance of routing protocol in MANET with combined scalable video coding. In: 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI). 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI). pp.1–4.

SHIVARAM, GUPTA, SHASHANK KAMATH, 2018. Queuing Models for Different Caching Schemes by Caching Partial Files. 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp.1234-1238

TANG, YIU, HUA., 2016. Exploit Every Bit: Effective Caching for High-Dimensional Nearest Neighbor Search. IEEE Transactions on

Knowledge and Data Engineering, 28(5), pp.1175-1188.

TANWIR, HENDRANTORO, G. & AFFANDI, A., 2015. Early result from adaptive combination of LRU, LFU and FIFO to improve cache server performance in telecommunication network. In: 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA). 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA). pp.429–432.

TANWIR, HENDRANTORO, G. & AFFANDI, A., 2017. Combination Of Fifo-Lru Cache Replacement Algorithms On Proxy Server To Improve Speed Of Response To Object Requests From Clients. ARPN Journal of Engineering and Applied Sciences, 12(3), p.6.

XIA, XIAO., 2016. High-Performance and Endurable Cache Management for Flash-Based Read Caching. IEEE Transactions on Parallel and Distributed Systems, 27(12)pp. 3518-3531

WANG, DONG, WU., 2017. Throughput Analysis of the Cache-Enabled Device-to-Device Communication and Small Base Stations Assisting in Cellular Networks. 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). p.414-421.

YEN, CHIEN, CHANG., 2018. Cooperative Online Caching in Small Cell Networks with Limited Cache Size and Unknown Content Popularity. 2018 3rd International Conference on Computer and Communication Systems (ICCCS), pp. 173-177.