Optimasi Kinerja Web Menggunakan *Application-Level Cache* di Sisi Server dan Browser

Widhiarta[1], Arief Setyanto[2], Ferry Wahyu Wibowo[3]
Magister Teknik Informatika, Universitas AMIKOM Yogyakarta
[1]greatguard9@gmail.com, [2]arief_s@amikom.ac.id, [3]ferry.w@amikom.ac.id

INTISARI

Penelitian ini bertujuan untuk melakukan optimasi kinerja web menggunakan application-level cache di sisi server dan browser. Penelitian ini disusun menggunakan 2 buah VPS 1 core, memori RAM 512MB, harddisk 40GB masing-masing untuk server web dan basis data, web server Apache 2.4 dengan PHP 7.1, basis data MariaDB v.10 dengan rekayasa 20 tabel dan 10 juta tupel.

Pengambilan sampel menggunakan perulangan 5x dengan kombinasi tingkat kueri dan tingkat konkurensi yang berbeda. Data dikumpulkan menggunakan aplikasi Apica Zebra Tester. Hasil analisis data menunjukkan kombinasi konfigurasi cache memiliki pengaruh yang berbeda terhadap kinerja web. Tanpa cache, kecepatan waktu akses web melambat drastis hingga 27.078,91 milidetik pada 50 konkurensi akses dan perulangan 100 kueri dengan hasil 100.000 data/kueri dengan jeda waktu 5 detik per konkurensi.

Hasil penelitian membuktikan bahwa konfigurasi cache di sisi browser memiliki pengaruh peningkatan kecepatan waktu akses rata-rata 79,61% dan penurunan beban CPU 80,83% tidak stabil ketika konkurensi akses dilakukan dengan profil browser berbeda. Konfigurasi cache di sisi server memiliki pengaruh peningkatan kecepatan waktu akses rata-rata 79,83% dan penurunan beban CPU 79,88%, stabil ketika konkurensi akses dilakukan dengan profil browser berbeda. Konfigurasi cache di sisi server dan browser memiliki peningkatan pengaruh kecepatan waktu akses rata-rata tertinggi 80,07% dan penurunan beban CPU tertinggi 82,64%, sangat stabil ketika konkurensi akses dilakukan dengan profil browser berbeda. Hasil uji membuktikan, konfigurasi application-level cache paling optimal menggunakan gabungan konfigurasi cache di sisi server dan browser.

Kata Kunci : *optimasi kinerja web, application-level cache, web cache, cache di sisi browser, cache di sisi server*

ABSTRACT

This research intends to optimizing web performance using application-level cache on server-side and browser-side. This research was arranged using 2 VPS with 1 core processor, 512MB RAM, 40GB SSD, Apache 2.4 web server with PHP 7.1, MariaDB v.10 database with 20 tables and 10 million tuples.

Sampling in this research using 5x loop with various query-level dan qonqurrency level.. Data were collected using Apica Zebra Tester application. Data analysis result shows the combination of cache configurations have different effects on web performance. Without cache, web access time speeds slowed dramatically to 27,078.91 milliseconds on 50 access concurrencies and 100 queries recurring with 100,000 data/query with of 5 seconds delay per concurrency.

The results show the browser-side cache configuration effect has 79,61% increasing response time access average and 80,83% decrease CPU load average, unstable when the concurrency access is done with different browser profiles. The server-side cache configuration effect has 79,83% increasing response time access average and 79,88% decrease CPU load average, stable when concurrency access is made with different browser profiles. The server-side and browser-side cache configuration effect has 80,07% increasing response time access average and 82,64% decrease CPU load average, very stable when concurrency access is performed with different browser profiles. The test results prove optimal application-level cache configuration uses a combination of server-side and browser-side.

Keyword : *web performance optimization, application-level cache, web cache, browser-side cache, server-side cache*

I. PENDAHULUAN

Sistem berbasis web telah digunakan secara luas dan dapat diakses oleh banyak orang melalui jaringan internet untuk berbagai macam kepentingan. Penyimpanan data dalam web yang semakin besar dan jumlah pengakses yang semakin banyak dapat menyebabkan kinerja web menurun. Ini adalah masalah yang umum terjadi pada sistem informasi yang melibatkan basis data. Kontrol Konkurensi dalam sistem basis data, yaitu proses pengaturan operasi–operasi dalam banyak transaksi yang berjalan secara simultan tanpa mengganggu operasi pada transaksi lainnya sehingga dapat menghasilkan data yang konsisten (Connolly, 2005, p577) didalamnya sangat bermanfaat untuk mengendalikan transaksi data sehingga antar proses dapat berjalan dengan lancar [2].

Penggunaan skrip yang bisa menjadi kompleks untuk mengambil data dari sistem basis data dan berulang dalam satuan waktu yang sangat singkat karena jumlah pengakses data bertambah dapat menyebabkan kemacetan proses karena kerapatan proses perintah lebih tinggi dibandingkan kemampuan server. Oleh karena itu perlu adanya penanganan khusus untuk meningkatkan kinerja web. Dalam penelitian ini diusulkan strategi *application-level cache* di sisi server dan browser untuk mengatasi beban kerja web.

Dari penelitian Chen, T.-H., Shang, W., Hassan, A. E., Nasser, M., & Flora, P. dengan judul CacheOptimizer: helping developers configure caching frameworks for hibernate-based database-centric web applications, disebutkan bahwa perlu bahasan penelitian tentang kontrol pengaturan arus informasi untuk optimalisasi konfigurasi *cache*, konfigurasi *application-level cache* dengan sifat *read-only* dan *read/write* untuk optimasi *cache* dan kontrol konkurensi *cache* dalam lingkungan terdistribusi [1]. Maka dalam penelitian ini, penulis melakukan konfigurasi *application-level cache* dengan sifat *read-only* di sisi browser klien dan *read/write* di sisi server.

Dari penelitian Liu, X., Ma, Y., Liu, Y., Xie, T., & Huang, G., dengan judul Demystifying the Imperfect Client-Side Cache Performance of Mobile Web Browsing memiliki fokus penelitian kinerja *cache* di sisi klien [5]. Penelitian menggunakan simulasi browser selular dengan browser chrome yang diinstal pada PC. Maka dalam penelitian ini, penulis melakukan penelitian pengaruh *cache* di sisi browser PC/laptop dan perangkat mobile.

Dari penelitian Mertz, J., & Nunes, I., dengan judul Seamless and Adaptive Application-level Caching, mengusulkan solusi desain kerangka kerja *cache* [9]. Maka dalam penelitian ini, penulis mengusulkan kerangka kerja dan perbandingan pengukuran hasil implementasi teknik *cache* dan tanpa *cache*.

Dari penelitian Nikolaou, S., Van Renesse, R., & Schiper, N., dengan judul Proactive Cache Placement on Cooperative Client Caches for Online Social Networks yang membahas peningkatan rasio klik *cache* individual dengan memanfaatkan hubungan antara klien memerlukan penelitian lanjutan tentang efisiensi *cache* yang ditempatkan di klien dengan adanya pergolakan klien, dampaknya terhadap latensi, biaya komputasi dan memori tambahan yang diperlukan untuk menjaga metadata *cache*, penelitian khusus tentang strategi gabungan skema *cache* proaktif dan skema *cache* populer, penelitian tentang kedekatan sosial apakah merugikan batas waktu bila dibandingkan dengan skema yang bergantung pada wilayah geografis [7]. Maka dalam penelitian ini, penulis melengkapi dengan teknik gabungan *cache* di server dan klien untuk mengatasi latensi dan biaya komputasi.

Dari penelitian Eyal, I., Birman, K., & Renesse, R. Van., dengan judul Cache Serializability: Reducing Inconsistency in Edge Transactions memerlukan perbaikan dengan kemungkinan mengubah ketergantungan objek secara dinamis dan mengizinkan aplikasi menginformasikan secara jelas ketergantungan *cache* dengan basis data [3]. Peneliti melakukan konfigurasi ketergantungan *cache* dinamis dengan basis data menggunakan pendekatan obyek id dan data *table*, sehingga *cache* merepresentasikan informasi terkini dari basis data.

Dari penelitian oleh Nanda, P., Singh, S., & Saini, G., dengan judul A Review of Web Caching Techniques and Caching Algorithms for Effective and Improved Caching General Terms disebutkan bahwa penelitian sistem *web cache* adaptif perlu dibangun menggunakan pembelajaran mesin untuk menyesuaikan diri dengan perubahan penting dalam pola penggunaan dan menyimpan objek yang sesuai di *cache* sambil memanfaatkan ruang *cache* secara efektif [6]. Dalam penelitian ini penulis memanfaatkan ruang

cache di sisi server dan klien secara lebih efektif sehingga hit rate menjadi lebih tinggi.

II. METODE PENELITIAN

2.1. Metode Penelitian

Proses interaksi program dengan basis data yang sangat rapat, jumlah kueri dan tingkat konkurensi akses dapat mempengaruhi kinerja web maka dirancang penelitian untuk melakukan optimasi menggunakan *application-level cache* di sisi server dan browser dengan metode eksperimental.

2.2. Metode Pengumpulan Data

Pengumpulan data dilakukan dengan melakukan percobaan penelitian kinerja aplikasi web mengggunakan Apica Zebra Tester. Alat ini digunakan untuk mengumpulkan data dan menguji kinerja web pada sistem operasi Centos 7.3. Pengujian dilakukan untuk mendapatkan data ukuran kecepatan waktu akses dan beban cpu yang terdiri dari penggunaan prosesor, penggunaan RAM dan beban I/O. Pengujian dilakukan menggunakan fitur *add on browser* Apica Zebra Tester yang dipasang pada browser Mozilla Firefox untuk melakukan pengujian secara nyata.

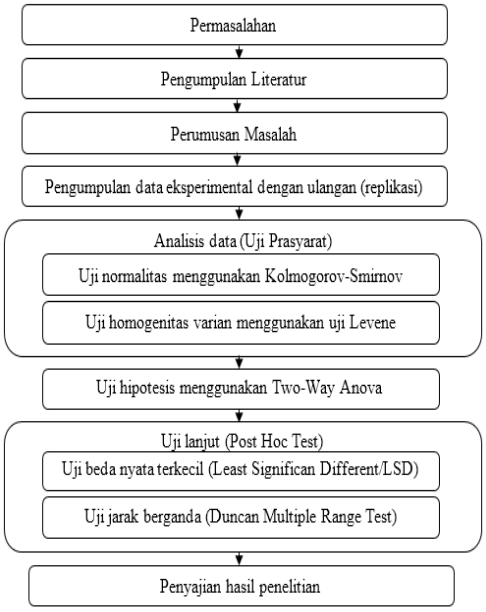
2.3. Metode Analisis Data

Setelah data didapatkan melalui aplikasi *Apica Zebra Tester* selanjutnya dilakukan analisis data sebagai berikut.

a) Penentuan variabel
b) Menyusun konfigurasi *cache*
c) Pengumpulan data menggunakan *Apica Zebra Tester*
d) Uji Normalitas *Kolmogorov-Smirnov*
e) Uji Homogenitas Varian *Levene*
f) Uji Hipotesis *two-way Anova*
g) Uji Beda Nyata Terkecil, *Least Significant Differences (LSD)*
h) Uji Jarak Berganda, *Duncan Multiple Range Test (DMRT)* untuk mengetahui jenis terbaik berdasarkan peringkat

2.4. Alur Penelitian

Alur penelitian ini menggunakan model penelitian eksperimental yang menghasilkan konfigurasi *application-level cache* terbaik untuk optimasi kinerja web. Tahap awal setelah mencari permasalahan, dilakukan pengumpulan literatur yang meliputi proses analisis metode, landasan teori dan analisis singkat tentang obyek penelitian. Selanjutnya melakukan perumusan masalah.



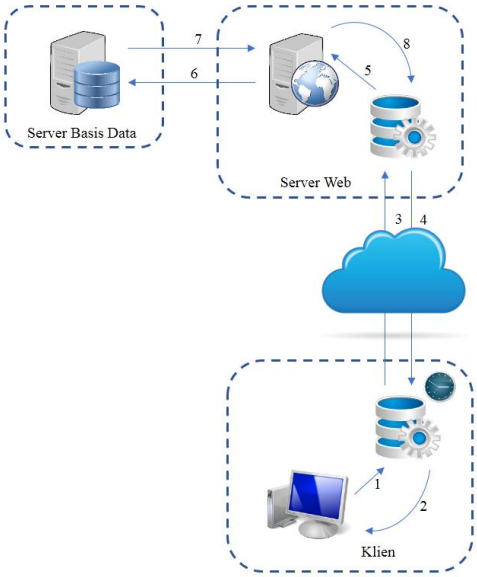
Gambar 1. Alur Penelitian

Adanya permasalahan kinerja web saat algoritma yang digunakan menjadi lebih kompleks, data yang diambil dari sistem basis data lebih banyak dan jumlah pengakses halaman web bertambah. Hal ini membuat halaman web menjadi semakin lambat diakses, bahkan tidak dapat diakses karena proses yang terjadi didalamnya menjadi semakin kompleks. Setelah masalah dirumuskan selanjutnya merancang desain arsitektur penelitian untuk mengatasi masalah tersebut.

Pada penelitian ini, dirancang 4 kondisi berbeda untuk mengetahui kinerja web paling optimal, yaitu kondisi tanpa *cache* (1), *cache* di sisi browser (2), *cache* di sisi server (3) serta *cache* di sisi server dan browser (4). Langkah selanjutnya adalah mengumpulkan data menggunakan *Apica Zebra Tester*.

Langkah selanjutnya melakukan analisis data dan menyajikan hasil penelitian.

Untuk mencari jawaban dari specific problem, maka terdapat langkah-langkah yang harus dilakukan antara lain adalah dengan melakukan konfigurasi eksperimental. Dengan menggunakan variabel jenis *cache*, kueri dan jumlah konkurensi akses. Server basis data, server web dan server testing terpisah. *Stress testing* menggunakan *Apica Zebra Tester* yang mampu digunakan untuk testing secara *real time* dan mendukung tes konkurensi akses serta mendukung *end-to-end testing* dengan *add on browser extension*. Berikut adalah desain arsitektur penelitian dan penjelasan sistem *application-level cache*.



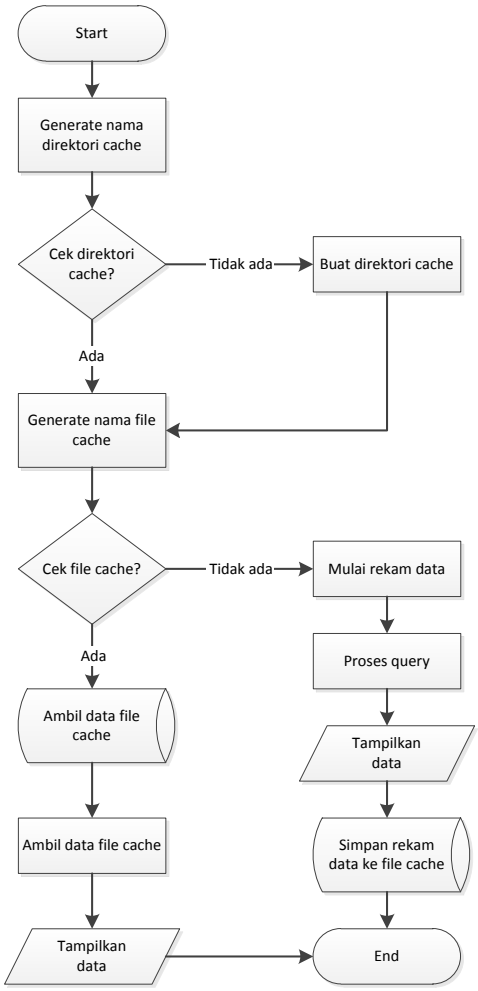
Gambar 2. Desain Arsitektur Penelitian

Proses kerja, pertama saat pengguna/klien melakukan permintaan informasi web menggunakan web browser, jika tersedia *cache* di browser lanjut ke proses no.2, jika tidak ada *cache*, proses lanjut ke no.3. Proses no.2, informasi web ditampilkan dalam browser. Proses no.3, permintaan informasi ke server web, jika tersedia *cache* di server web, lanjut ke proses no.4, jika tidak ada *cache*, lanjut ke proses no.5. Proses no.4, informasi web dikirimkan dari server web ke klien, lanjut ke proses no.2. Proses no.5, aplikasi web di server memproses atas permintaan informasi, lanjut ke proses no.6. Proses no.6, aplikasi web melakukan proses permintaan informasi ke server basis data, lanjut ke proses no.7. Proses no.7, server basis data memberikan hasil informasi ke aplikasi server web, lanjut ke proses no.8. Proses no.8, aplikasi web di server melakukan pengolahan sehingga informasi dapat dikirim ke klien, lanjut ke proses no.4.

III. HASIL ANALISIS DAN PEMBAHASAN

3.1. Konfigurasi Percobaan

Untuk melakukan percobaan penelitian eksperimental, maka dibuat rancangan flowchart sistem algoritma *cache* yang berguna untuk melakukan optimasi kinerja web sebagai berikut.



Gambar 3. Flowchart *application-level cache*

Selanjutnya adalah menggunakan modifikasi kueri untuk menampilkan data pada halaman web. Contoh kueri yang digunakan dalam percobaan dengan **i** adalah nilai perulangan yang dilakukan untuk masing-masing tingkat kueri dan **j** adalah nilai data yang ditampilkan, sebagai berikut.

"SELECT p.id_pasien, p.kode_pasien, p.nama_pasien, p.tempat_lahir, p.tgl_lahir, p.jen_kelamin, k.nama_pekerjaan, p.alamat, p.telp1, p.telp2, p.email, p.gol_darah, p.sakit_jantung, p.diabetes, p.haemopolia, p.hepatitis, p.gastring, p.alergi, p.alergi_pada, p.foto, p.status_member, p.status_cicilan, p.poin_aktif, p.poin_nonaktif, p.poin_transaksi, s.nama_instansi, c.nama_cabang, p.tgl_kedatangan, p.tgl_input, p.tgl_edit, p.status, m.nama_member FROM pasien p, pekerjaan k, paskerjasama_kategori s, cabang c, m.nama_member WHERE p.id_pekerjaan =

*k.id_pekerjaan AND p.id_paskerjasama_kategori = s.id_paskerjasama_kategori AND p.id_cabang = c.id_cabang AND p.id_member = m.id_member limit **i**,**j**".*

Kombinasi nilai dan **i** dan **j** diatur dan dilakukan tes untuk mendapatkan data uji kueri dan perulangan maksimal saat kondisi tanpa *cache* yang dapat diproses oleh server. Berikut adalah konfigurasi percobaan dengan tingkat kueri yang berbeda.

- **K0** = **Kueri ringan** dengan perulangan **i** dilakukan 1x
- **K1** = **Kueri sedang** dengan **i** adalah nilai perulangan kueri 10x dengan 5x nilai **i** yang sama.
- **K2** = **Kueri berat** dengan **i** adalah nilai perulangan kueri 100x dengan 50x nilai **i** yang sama.

Tingkat konkurensi:
- **A0** = Tanpa konkurensi dengan jumlah akses 1
- **A1** = Konkurensi dengan jumlah akses 10
- **A2** = Konkurensi dengan jumlah akses 50

Berikut adalah konfigurasi untuk masing-masing kelompok kondisi.

1. Konfigurasi tanpa *cache*

Berikut adalah konfigurasi pada file htaccess yang digunakan untuk menolak penyimpanan *cache* pada browser.

```
<IfModule mod_rewrite.c>
RewriteEngine on
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . / index.php [L]
</IfModule>
<ifModule mod_deflate.c>
  <filesMatch "\.(css|js|x?html?|php)$">
    SetOutputFilter DEFLATE
  </filesMatch>
</ifModule>
<FilesMatch "\.(html|htm|js|css|php)>
FileETag None
Header unset ETag
Header set Cache-Control "max-age=0, no-cache, no-store, must-revalidate"
Header set Pragma "no-cache"
Header set Expires "Wed, 11 Jan 1984 05:00:00 GMT"
</FilesMatch>
<IfModule mod_headers.c>
Header set Connection keep-alive
</IfModule>
```

2. Konfigurasi *cache* di sisi browser

Berikut adalah konfigurasi pada file htaccess yang digunakan untuk melakukan penyimpanan *cache* pada browser.

```
<IfModule mod_rewrite.c>
RewriteEngine on
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . / index.php [L]
</IfModule>
<ifModule mod_deflate.c>
  <filesMatch "\.(css|js|x?html?|php)$">
    SetOutputFilter DEFLATE
  </filesMatch>
</ifModule>
<IfModule mod_expires.c>
# Enable expirations
ExpiresActive On
# Default directive
ExpiresDefault "access plus 1 month"
# My favicon
ExpiresByType image/x-icon "access plus 1 year"
# Images
ExpiresByType image/gif "access plus 1 month"
ExpiresByType image/png "access plus 1 month"
ExpiresByType image/jpg "access plus 1 month"
ExpiresByType image/jpeg "access plus 1 month"
# CSS
ExpiresByType text/css "access 1 month"
# Javascript
ExpiresByType application/javascript "access plus 1 year"
</IfModule>
<IfModule mod_headers.c>
Header set Connection keep-alive
</IfModule>
```

3. Konfigurasi *Cache* di sisi server

Untuk cache di sisi server, konfigurasi file .htaccess menggunakan konfigurasi file .htaccess no.1 (tanpa *cache*) dan konfigurasi skrip php pada aplikasi web yang digunakan untuk menyimpan data di sisi server. Berikut

adalah konfigurasi skrip pada pada aplikasi web yang digunakan untuk melakukan penyimpanan *cache* di sisi server. Konfigurasi terbagi menjadi 3 file.

File top-cache.php berguna untuk memulai merekam *cache* satu halaman web utuh dengan ekstensi file .cch jika *cache* belum terbentuk. Jika *cache* sudah terbentuk, maka akan diambil data *cache* dari direktori server dan menghentikan proses eksekusi skrip berikutnya.

```
<?php
$cachedir = 'cache';
$filecache = '';
if (!file_exists($cachedir)) {
  mkdir($cachedir, 0755, true);
}
$urlcache1= print_r($_REQUEST, true);
$filecache = md5($urlcache.$urlcache1);
$cache_ext = '.cch';
$cachefile=$cachedir.'/cch-
'.seourl($filecache).$cache_ext;
if (file_exists($cachefile)) {
   ob_start();
   include($cachefile);
   ob_end_flush();
   exit();
}
?>
```

File bottom-cache.php digunakan untuk menyimpan rekaman halaman web utuh untuk disimpan dalam *cache* server.

```
<?php
$cached = fopen($cachefile, 'w');
fwrite($cached, ob_get_contents());
fclose($cached);
ob_end_flush();
?>
```

Skrip yang berguna untuk menyimpan data obyek hasil kueri basis data sehingga jika terjadi kueri yang sama, data akan diambil dari hasil kueri yang telah disimpan sebelumnya dengan ekstensi file *cache* .obj.

```
<?php $cache_obj = md5($sql);
$obj_file='cache/cch-
'.seourl($cache_obj).'.obj';
if(file_exists($obj_file)) {
$data .= file_get_contents($obj_file);
}
?>
```

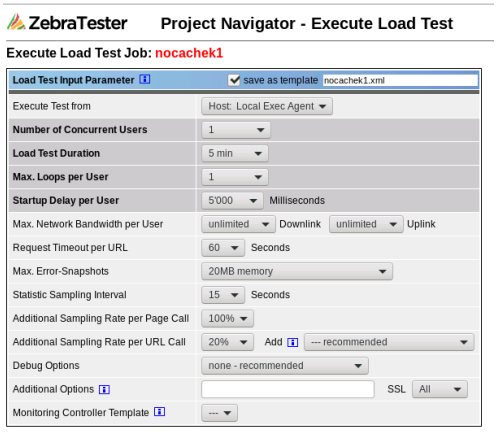
Jika data obyek *cache* belum ada pada direktori *cache*, maka perintah kueri akan dieksekusi dan disimpan dengan skrip berikut.

```
<?php
$myfile = fopen($obj_file, "w");
fwrite($myfile, $obj_data);
fclose($myfile);
?>
```

4. Konfigurasi *Cache* di sisi server dan browser

Cache di sisi server dan browser menggunakan gabungan konfigurasi no.2 file .htaccess di sisi browser untuk menyimpan *cache* di browser dan konfigurasi file top-cache.php, bottom-cache.php dan skrip untuk menyimpan obyek data menjadi *cache* di sisi server.

Berikut adalah konfigurasi percobaan menggunakan *Apica Zebra Tester*.



Gambar 4. Konfigurasi *Apica Zebra Tester*

Keterangan konfigurasi:

1. *Execute test from: Host local exec agent*
2. *Number of concurrent users: 50*
3. *Load test duration: 5 min*
4. *Max. Loops per user: 50*
5. *Startup delay per user: 5.000 ms*
6. *Max. Network bandwidth per user: unlimited*
7. *Request timeout per URL: 60 seconds*
8. *Max. Error snapshots: 20MB memory*
9. *Statistic sampling interval: 15 seconds*
10. *Additional sampling rate per page call: 100%*
11. *Additional sampling rate per URL call: 20%*
12. *Debug options: none - recommended*
13. *Additional options: -*
14. *Monitoring controller template: -*

Hasil percobaan awal yang dilakukan sebanyak 5x perulangan (U) untuk kondisi tanpa *cache*, aplikasi web dapat diakses dengan hasil kecepatan waktu akses tertinggi 27.078,91 milidetik dengan CPU *overload* hingga 9,38 untuk konkurensi akses 50 pengguna (A2) untuk kueri berat 100 perulangan dengan hasil 100.000 data per kueri (K2).

TABEL I.
DATA PERCOBAAN KONDISI TANPA *CACHE*

C	K	U	HK (ms)	HC
C0	K2	1	19.685,39	4,45
C0	K2	2	21.585,65	6,56
C0	K2	3	27.583,06	7,86
C0	K2	4	25.495,36	8,76
C0	K2	5	27.078,91	9,38

3.2. Analisis Data

1. Uji Normalitas *Kolmogorov-Smirnov*

Untuk mengethaui normal atau tidaknya distribusi data, maka dilakukan uji normalitas. Uji beda antara data yang diuji normalitasnya dengan data normal baku menggunakan uji *Kolmogorov-Smirnov*. Konsep dasar dari uji normalitas *Kolmogorov-Smirnov* adalah dengan membandingkan distribusi data (yang akan diuji normalitasnya) dengan distribusi normal baku. Distribusi normal baku adalah data yang telah ditransformasikan ke dalam bentuk *Z-score* dan diasumsikan normal. Jika signifikansi p < 0,05 berarti terdapat perbedaan yang signifikan, dan jika signifikansi p > 0,05 maka tidak terjadi perbedaan yang signifikan (data terdistribusi normal).

Berikut adalah tabel uji normalitas kecepatan waktu akses untuk 50 konkurensi akses untuk masing-masing kondisi *cache*.

TABEL II.
UJI NORMALITAS KECEPATAN WAKTU AKSES 50 KONKURENSI AKSES UNTUK KONDISI *CACHE*

C	Z K-S	p	Status
C0	0,788	0,563	Normal
C1	1,156	0,138	Normal
C2	1,128	0,157	Normal
C3	1,098	0,180	Normal

Berikut adalah tabel uji normalitas kecepatan waktu akses untuk 50 konkurensi akses untuk masing-masing tingkat kueri.

TABEL III.
UJI NORMALITAS KECEPATAN WAKTU AKSES 50 KONKURENSI AKSES UNTUK TINGKAT KUERI

K	Z K-S	p	Status
K0	1,582	0,013	Tidak Normal
K1	2,019	0,001	Tidak Normal
K2	2,031	0,001	Tidak Normal

Berikut adalah tabel uji normalitas penggunaan beban CPU untuk 50 konkurensi akses untuk masing-masing kondisi *cache*.

TABEL IV.
UJI NORMALITAS PENGGUNAAN BEBAN CPU 50 KONKURENSI AKSES UNTUK KONDISI *CACHE*

C	Z K-S	p	Status
C0	0,771	0,592	Normal
C1	1,713	0,006	Tidak Normal
C2	1,526	0,019	Tidak Normal
C3	1,564	0,015	Tidak Normal

Berikut adalah tabel uji normalitas penggunaan beban CPU untuk 50 konkurensi akses untuk masing-masing tingkat kueri.

TABEL V.
UJI NORMALITAS PENGGUNAAN BEBAN CPU 50 KONKURENSI AKSES UNTUK TINGKAT KUERI

K	Z K-S	P	Status
K0	1,524	0,019	Tidak Normal
K1	1,602	0,012	Tidak Normal
K2	1,655	0,008	Tidak Normal

Hasil uji normalitas kecepatan waktu akses untuk kondisi *cache* pada tabel 2. menunjukkan status sebaran normal. Hal ini membuktikan bahwa masing-masing kondisi *cache* memiliki konsistensi waktu akses yang stabil.

Hasil uji normalitas kecepatan waktu akses untuk tingkat kueri pada tabel 3. menunjukkan status sebaran tidak normal. Hal ini membuktikan bahwa masing-masing tingkat kueri memiliki waktu akses yang tidak stabil.

Hasil uji normalitas penggunaan beban CPU untuk kondisi *cache* pada tabel 4. menunjukkan status sebaran tidak normal. Hal ini membuktikan bahwa masing-masing kondisi *cache* menyebabkan penggunaan beban CPU yang tidak stabil.

Hasil uji normalitas penggunaan beban CPU untuk tingkat kueri pada tabel 5. menunjukkan status sebaran tidak normal. Hal ini membuktikan bahwa masing-masing tingkat kueri menyebabkan penggunaan beban CPU yang tidak stabil.

2. Uji Homogenitas Varian *Levene*

Uji homogenitas varian dimaksudkan untuk menguji apakah varian antar kelompok perlakuan bersifat homogen atau tidak. Jika signifikansi p > 0,05 pada uji levene, maka dapat disimpulkan bahwa varian antar kelompok bersifat homogen.

Berikut adalah tabel uji homogenitas varian kecepatan waktu akses untuk 50 konkurensi akses.

TABEL VI.
UJI LEVENE PERCOBAAN 50 KONKURENSI AKSES TERHADAP KECEPATAN WAKTU AKSES

Grup	F-score	p	Status
Cache	25,917	0,000	Tidak Homogen
Kueri	27,283	0,000	Tidak Homogen

Berikut adalah tabel uji homogenitas varian penggunaan beban CPU untuk 50 konkurensi akses.

TABEL VII.
UJI LEVENE PERCOBAAN 50 KONKURENSI AKSES TERHADAP PENGGUNAAN BEBAN CPU

Grup	F-score	p	Status
Cache	40,424	0,000	Tidak Homogen
Kueri	14,364	0,000	Tidak Homogen

Hasil uji homogenitas varian pada tabel 6 menunjukkan bahwa status varian bersifat tidak homogen. Hal ini karena terdapat perbedaan kecepatan waktu akses yang bervariasi untuk masing-masing kondisi *cache* maupun tingkat kueri.

Hasil uji homogenitas varian pada tabel 7 menunjukkan bahwa status varian bersifat tidak homogen. Hal ini karena terdapat perbedaan penggunaan beban CPU yang bervariasi untuk masing-masing tingkat *cache* maupun tingkat kueri.

3. Uji Hipotesis *two-way* Anova

Anova (*Analysis of Varian*) adalah uji komparatif yang digunakan untuk menguji perbedaan *mean* (rata-rata) data lebih dari dua kelompok. Uji hipotesis dilakukan menggunakan *Univariate Two Way Analysis of Variance* dengan variabel bebas ada 2, sedangkan variabel terikat ada satu.

Berikut adalah tabel uji hipotesis kecepatan waktu akses untuk 50 konkurensi akses.

TABEL VIII.
UJI HIPOTESIS KECEPATAN WAKTU AKSES PERCOBAAN 50 KONKURENSI AKSES

Grup	F-score	p	Status
Cache	574,932	0,000	Bermakna
Kueri	149,296	0,000	Bermakna
Interaksi C*K	149,723	0,000	Bermakna

Berikut adalah tabel uji hipotesis penggunaan beban CPU untuk 50 konkurensi akses.

TABEL IX.
UJI HIPOTESIS PENGGUNAAN BEBAN CPU PERCOBAAN 50 KONKURENSI AKSES

Grup	F-score	p	Status
Cache	113,307	0,000	Bermakna
Kueri	21,297	0,000	Bermakna
Interaksi C*K	19,772	0,000	Bermakna

Hasil uji hipotesis pada tabel 8 menunjukkan bahwa status kelompok hipotesis kecepatan waktu akses bermakna. Hal ini karena masing-masing kondisi *cache*, tingkat kueri dan interaksi antara *cache* dan kueri menyebabkan perbedaan kecepatan waktu akses.

Hasil uji hipotesis pada tabel 9 menunjukkan bahwa status kelompok penggunaan beban CPU bermakna. Hal ini karena masing-masing kondisi *cache*, tingkat kueri dan interaksi antara *cache* dan kueri menyebabkan perbedaan penggunaan beban CPU.

4. Uji Beda Nyata Terkecil *Least Significant Differences (LSD)*

Uji Beda Nyata Terkecil atau *LSD test* (*Least Significant Differences*) juga dikenal dengan Uji T berganda (*Multiple T Test*) adalah metode yang diperkenalkan oleh Ronald Fisher. Metode ini menjadikan nilai LSD sebagai acuan dalam menentukan apakah rata-rata dua perlakuan berbeda secara statistik atau tidak.

Berikut adalah tabel uji LSD kecepatan waktu akses antar kelompok *cache* untuk 50 konkurensi akses.

TABEL X.
UJI LSD PERCOBAAN 50 KONKURENSI AKSES ANTAR KELOMPOK *CACHE* TERHADAP KECEPATAN WAKTU AKSES

Grup	p	Keterangan
C0*C1	0,000	Bermakna
C0*C2	0,000	Bermakna
C0*C3	0,000	Bermakna
C1*C2	0,999	Tidak bermakna
C1*C3	0,938	Tidak bermakna
C2*C3	0,939	Tidak bermakna

Berikut adalah tabel uji LSD penggunaan beban CPU antar kelompok *cache* untuk 50 konkurensi akses.

TABEL XI.
UJI LSD PERCOBAAN 50 KONKURENSI AKSES ANTAR KELOMPOK *CACHE* TERHADAP BEBAN CPU

Grup	p	Keterangan
C0*C1	0,000	Bermakna
C0*C2	0,000	Bermakna
C0*C3	0,000	Bermakna
C1*C2	0,973	Tidak bermakna
C1*C3	0,841	Tidak bermakna
C2*C3	0,815	Tidak bermakna

Berikut adalah tabel uji LSD kecepatan waktu akses antar kelompok kueri untuk 50 konkurensi akses.

TABEL XII.
UJI LSD PERCOBAAN 50 KONKURENSI AKSES ANTAR KELOMPOK KUERI TERHADAP KECEPATAN WAKTU AKSES

Grup	P	Keterangan
K0*K1	0,000	Bermakna
K0*K2	0,000	Bermakna
K1*K2	0,000	Bermakna

Berikut adalah tabel uji LSD penggunaan beban CPU antar kelompok kueri untuk 50 konkurensi akses.

TABEL XIII.
UJI LSD 50 KONKURENSI AKSES ANTAR KELOMPOK KUERI TERHADAP BEBAN CPU

Grup	p	Keterangan
K0*K1	0,000	Bermakna
K0*K2	0,000	Bermakna
K1*K2	0,115	Tidak Bermakna

Hasil uji LSD pada tabel 10 menunjukkan bahwa kecepatan waktu akses antar kelompok *cache* C0*C1, C0*C2, C0*C3 bermakna, sedangkan untuk C1*C2, C1*C3 dan C2*C3 tidak bermakna. Hal ini karena kondisi tanpa *cache* (C0) memiliki perbedaan kecepatan waktu akses yang tinggi dibandingkan kondisi dengan *cache* di sisi browser (C1), *cache* di sisi server (C2) dan *cache* di sisi server dan browser (C3). Sedangkan perbedaan kecepatan waktu akses antar kondisi dengan *cache* tidak terlalu signifikan.

Hasil uji LSD pada tabel 11 menunjukkan bahwa penggunaan beban CPU antar kelompok *cache* C0*C1, C0*C2, C0*C3 bermakna, sedangkan untuk C1*C2, C1*C3 dan C2*C3 tidak bermakna. Hal ini karena kondisi tanpa *cache* (C0) memiliki perbedaan penggunaan beban CPU yang tinggi dibandingkan kondisi dengan *cache* di sisi browser (C1), *cache* di sisi server (C2) dan *cache* di sisi server dan browser (C3). Sedangkan perbedaan penggunaan beban CPU antar kondisi dengan *cache* tidak terlalu signifikan.

Hasil uji LSD pada tabel 12 menunjukkan bahwa kecepatan waktu akses antar kelompok kueri K0*K1, K0*K2, K1*K2 bermakna. Hal ini karena semua tingkat kueri memiliki tingkat kueri tinggi sehingga menyebabkan perbedaan kecepatan waktu akses yang tinggi.

Hasil uji LSD pada tabel 13 menunjukkan bahwa penggunaan beban CPU antar kelompok kueri K0*K1, K0*K2 bermakna sedangkan antar K1*K2 tidak bermakna. Hal ini karena tingkat kueri rendah (K0) dan sedang (K1) memiliki perbedaan tingkat kueri yang tinggi, sedangkan antar K1 dan K2 tidak terlalu signifikan.

5. Uji Jarak Berganda *Duncan Multiple Range Test (DMRT)*

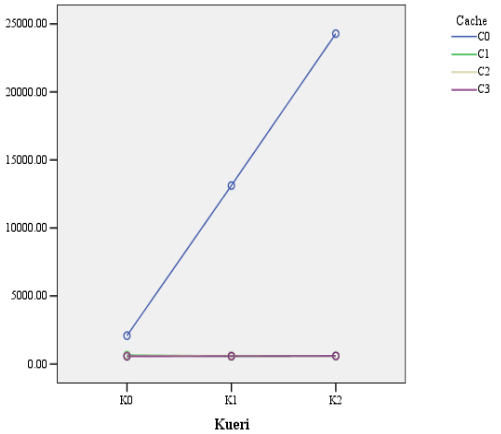
Uji Jarak Ganda Duncan atau Uji DMRT (*Duncan Multiple Range Test*) digunakan untuk mengetahui jenis terbaik berdasarkan rankingnya. Uji ini dilakukan karena adanya perbedaan nyata pada hasil analisis varian.

Berikut adalah tabel uji DMRT kecepatan waktu akses kelompok *cache* untuk 50 konkurensi akses.

TABEL XIV.
UJI DMRT PERCOBAAN 50 KONKURENSI AKSES KELOMPOK *CACHE* TERHADAP KECEPATAN WAKTU AKSES

Kelompok	Grup 1	Grup 2
C0		13.162,46
C1	607,4413	
C2	606,8860	
C3	578,4473	

Perbedaan nyata antar grup ditunjukkan dalam grafik pada gambar 5 berikut.



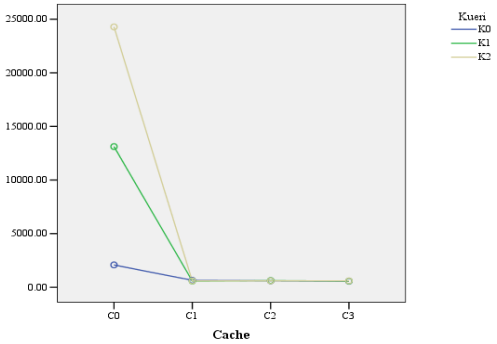
Gambar 5. Grafik uji DMRT percobaan 50 konkurensi akses kelompok *cache* terhadap kecepatan waktu akses

Berikut adalah tabel uji DMRT kecepatan waktu akses kelompok kueri untuk 50 konkurensi akses.

TABEL XV.
UJI DMRT PERCOBAAN 50 KONKURENSI AKSES KELOMPOK KUERI TERHADAP KECEPATAN WAKTU AKSES

Grup	Grup 1	Grup 2	Grup 3
K0	973,2215		
K1		3.724,9845	
K2			6.518,2175

Perbedaan nyata antar grup ditunjukkan dalam grafik pada gambar 6 berikut.



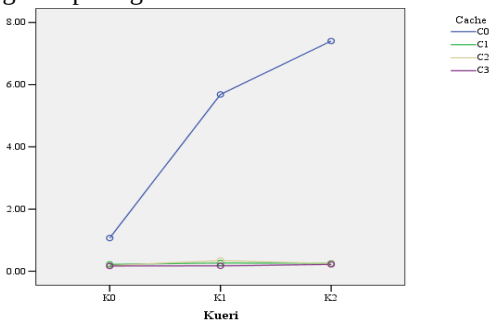
Gambar 6. Grafik uji DMRT percobaan 50 konkurensi akses kelompok kueri terhadap kecepatan waktu akses

Berikut adalah tabel uji DMRT penggunaan beban CPU kelompok *cache* untuk 50 konkurensi akses.

TABEL XVI.
UJI DMRT PERCOBAAN 50 KONKURENSI AKSES KELOMPOK *CACHE* TERHADAP PENGGUNAAN BEBAN CPU

Kelompok	Grup 1	Grup 2
C0		4,7187
C1	0,2513	
C2	0,2613	
C3	0,1913	

Perbedaan nyata antar grup ditunjukkan dalam grafik pada gambar 7 berikut.



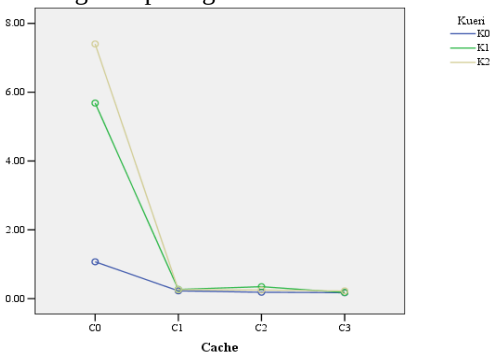
Gambar 7. Grafik uji DMRT percobaan 50 konkurensi akses kelompok *cache* terhadap penggunaan beban CPU

Berikut adalah tabel uji DMRT penggunaan beban CPU kelompok kueri untuk 50 konkurensi akses.

TABEL XVII.
UJI DMRT PERCOBAAN 50 KONKURENSI AKSES KELOMPOK KUERI TERHADAP PENGGUNAAN BEBAN CPU

Kelompok	Grup 1	Grup 2
K0	0,4135	
K1		1,6195
K2		2,0340

Perbedaan nyata antar grup ditunjukkan dalam grafik pada gambar 8 berikut.



Gambar 8. Grafik uji DMRT percobaan 50 konkurensi akses kelompok kueri terhadap penggunaan beban CPU

3.3. Analisis Data

Berdasarkan data primer yang didapatkan dari aplikasi Apica Zebra Tester dapat dihitung pengaruh kinerja web yang terbagi menjadi 2 yaitu, kecepatan waktu akses dan beban CPU dengan rumus persentase perbandingan sebagai berikut.

$$HKp = \frac{(\bar{X}HK0 - \bar{X}HKn)}{\bar{X}HK0} \; x \; 100\%$$

$$HCp = \frac{(\bar{X}HC0 - \bar{X}HCn)}{\bar{X}HC0} \; x \; 100\%$$

Keterangan:

HKp = Hasil persentase perbandingan peningkatan kecepatan waktu akses

$\bar{X}HK0$ = Rerata kecepatan waktu akses kelompok tanpa *cache*

$\bar{X}HKn$ = Rerata kecepatan waktu akses kelompok n

HCp = Hasil persentase perbandingan penurunan beban CPU

$\bar{X}HC0$ = Rerata beban CPU kelompok tanpa *cache*

$\bar{X}HCn$ = Rerata beban CPU kelompok n

Berikut adalah pengaruh perbandingan peningkatan kecepatan waktu akses percobaan

50 konkurensi akses yang ditunjukkan pada tabel XVIII.

TABEL XVIII.
PENGARUH PERBANDINGAN PENINGKATAN KECEPATAN WAKTU AKSES PERCOBAAN 50 KONKURENSI AKSES

C \ K	K0	K1	K2
C0	0%	0%	0%
C1	68,88%	95,58%	97,55%
C2	71,16%	95,24%	97,55%
C3	72,98%	95,61%	97,54%

Berikut adalah pengaruh perbandingan peningkatan kecepatan waktu akses percobaan 50 konkurensi akses yang ditunjukkan pada tabel XIX.

TABEL XIX.
PENGARUH PERBANDINGAN PENURUNAN BEBAN CPU PERCOBAAN 50 KONKURENSI AKSES

C \ K	K0	K1	K2
C0	0%	0%	0%
C1	78,88%	92,29%	96,49%
C2	82,62%	93,88%	96,62%
C3	83,93%	96,87%	96,97%

Berdasarkan tabel 18 dan tabel 19, penggunaan *cache* di sisi browser, *cache* di sisi server dan *cache* di sisi server dan browser pada kueri sedang (K1) dan kueri berat (K2) memiliki hasil peningkatan kecepatan waktu akses dan penurunan beban CPU yang hampir sama. Hal ini dikarenakan aplikasi *Apica Zebra Tester* tidak memiliki kemampuan untuk membedakan profil browser berbeda saat simulasi *stress testing* dilakukan dalam waktu bersamaan.

Percobaan pembedaan profil browser hanya dapat dilakukan dengan waktu terpisah. Setelah percobaan selesai, dilakukan uji coba ulang dengan cara menutup aplikasi dan membuka ulang aplikasi *Apica Zebra Tester* dengan profil browser baru menggunakan *add-on browser multifox* untuk mengabaikan *cache* browser sebelumnya. Hasilnya, kondisi *cache* di sisi browser (C1) membutuhkan waktu akses lebih lama saat awal akses dibandingkan dengan *cache* di sisi server (C2) dan *cache* di sisi server dan browser (C3). Hal ini karena tidak ada data *cache* yang ditemukan di sisi browser, sedangkan *cache* di sisi server (C2) dan *cache* di sisi server dan browser (C3) masih memiliki *cache* di sisi server, sehingga dapat diakses lebih cepat.

Penggunaan *cache* di sisi browser mampu meningkatkan kecepatan waktu akses khusus untuk profil browser tersebut. Hal ini mengakibatkan *cache* di sisi browser tidak mampu mengatasi kinerja web saat halaman web diakses oleh banyak pengguna dengan lokasi dan atau profil browser yang berbeda. Hal ini akan menyebabkan beban CPU menjadi tinggi dan dapat menyebabkan *overload* sehingga halaman web lama atau bahkan tidak dapat diakses.

Penggunaan *cache* di sisi server mampu meningkatkan kecepatan waktu akses dan meringankan beban CPU saat diakses oleh banyak pengguna dengan lokasi dan atau profil browser berbeda. Hal ini karena *cache* di simpan di sisi server dan dapat digunakan oleh banyak pengguna di lokasi dan atau profil browser berbeda secara bersama.

Perbedaan terjadi saat pengguna yang sama mengakses halaman web yang sama dengan profil browser tetap. *Cache* di sisi browser tidak perlu melakukan permintaan halaman web melalui jaringan kembali, karena saat permintaan dilakukan, browser akan melakukan cek ketersediaan data *cache* yang di minta. Jika data *cache* masih ada, maka halaman web dapat langsung ditampilkan oleh browser. Sedangkan *cache* di sisi server akan selalu meminta halaman web melalui jaringan. Hal ini dapat menyebabkan perbedaan kecepatan waktu akses jika bandwidth jaringan tidak stabil.

Penelitian oleh Chen, T.-H., Shang, W., Hassan, A. E., Nasser, M., & Flora, P., dalam International Symposium on the Foundations of Software Engineering (pp. 666–677), 2016, dengan judul CacheOptimizer: helping developers configure caching frameworks for hibernate-based database-centric web applications yang fokus pada *cache* berdasarkan *web log* membuktikan bahwa *cache* di sisi server mampu mengatasi kinerja web dan meningkatkan throughput sebesar 27% (Pet Clinic), 138% (Cloud Store), 45% (OpenMRS) [1].

Penelitian oleh Liu, X., Ma, Y., Liu, Y., Xie, T., & Huang, G., dalam IEEE Transactions on Mobile Computing, 15(9), 2206–2220, 2016, dengan judul Demystifying the Imperfect Client-Side Cache Performance of Mobile Web Browsing yang fokus pada *cache* di sisi browser menyatakan bahwa tidak semua *cache* browser bekerja dengan baik, terutama pada perangkat *mobile* [5]. *Cache* di sisi browser sangat berguna untuk menghemat bandwidth jaringan, sehingga browser tidak perlu melakukan permintaan yang sama ke

server untuk konten yang sama sehingga mampu jeda waktu akses dapat diminimalisir.

Penelitian oleh Mertz, J., & Nunes, I., dalam VI Workshop de Teses e Dissertações do CBSoft (WTDSoft 2016) (p. 7), 2016, dengan judul Seamless and Adaptive Application-level Caching yang mengusulkan *application-level cache* yang adaptif dan dapat diintegrasikan pada aplikasi web menyatakan bahwa obyek data dalam aplikasi web perlu di simpan menjadi *cache*, sehingga data dapat ditampilkan atau diproses lebih cepat tanpa harus mengambil data dari sistem basis data [9].

Penelitian oleh Eyal, I., Birman, K., & Renesse, R. Van., dalam International Conference on Distributed Computing Systems (Vol. 2015–July, pp. 686–695), 2015 dengan judul Cache Serializability: Reducing Inconsistency in Edge Transactions yang mengusulkan *T-Cache* untuk meningkatkan konsistensi *cache* dengan basis data dengan *cache* terpusat untuk menangani basis data pada fungsi transaksi juga mengandalkan *cache* di sisi server, karena data khususnya data transaksi harus dapat diolah lebih cepat dan menunjukkan data terkini [3].

Penelitian oleh Nanda, P., Singh, S., & Saini, G., dalam International Journal of Computer Applications, 128(10), 975–8887, 2015 dengan judul A Review of Web Caching Techniques and Caching Algorithms for Effective and Improved Caching General Terms membahas tentang arsitektur *cache* meliputi *cache* proxy, *cache* kooperatif, *cache* adaptif, *cache* push dan *cache* aktif [6]. *Web cache* adalah solusi yang paling sesuai dan paling berkelanjutan untuk mengurangi lalu lintas internet dan konsumsi bandwidth.

Berdasarkan penelitian sebelumnya, terbukti bahwa *cache* di server dapat digunakan untuk meringankan beban CPU dan *cache* di sisi browser dapat digunakan untuk menghemat koneksi dan mempercepat waktu akses untuk konten yang sama. Oleh karena itu, dalam penelitian ini terbukti bahwa konfigurasi paling optimal adalah dengan kombinasi *cache* di sisi server dan browser. Dengan kombinasi ini, pengguna akan dapat mengakses halaman web lebih cepat karena *cache* dapat tersimpan di sisi server maupun browser.

IV. KESIMPULAN

Dari hasil analisis dan pembahasan diperoleh kesimpulan sebagai berikut.

Penggunaan *cache* mampu meringankan beban CPU dan meningkatkan kecepatan akses halaman web.

Hasil penelitian membuktikan bahwa konfigurasi *cache* di sisi browser meningkatkan kecepatan waktu akses rata-rata sebesar 79,61% dan penurunan beban CPU sebesar 80,83% dalam 1 profil browser. Konfigurasi *cache* di sisi server meningkatkan kecepatan waktu akses rata-rata sebesar 79,83% dan menurunkan beban CPU sebesar 79,88% dan dapat diakses dengan profil browser berbeda. Konfigurasi *cache* di sisi server dan browser meningkatkan kecepatan waktu akses rata-rata sebesar 80,07% dan menurunkan beban CPU sebesar 82,64% dan stabil ringan saat diakses oleh pengguna melalui lokasi atau profil browser berbeda.

Penggunaan paling optimal adalah dengan kombinasi *cache* di sisi server dan browser. Dengan kombinasi ini, pengguna akan dapat mengakses halaman web lebih cepat karena *cache* dapat tersimpan di sisi server maupun browser.

Penelitian ini dapat dikembangkan untuk web yang lebih kompleks seperti forum, transaksi, portal, *e-learning* atau web dinamis lain yang memiliki konten besar dan jumlah pengakses besar.

Penggunaan aplikasi *web stress tester* lain kemungkinan akan didapatkan data yang lebih variatif.

REFERENSI

[1] Chen, T.-H., Shang, W., Hassan, A. E., Nasser, M., & Flora, P. 2016. CacheOptimizer: Helping Developers Configure Caching Frameworks for Hibernate-based Database-centric Web Applications. In *International Symposium on the Foundations of Software Engineering* (pp. 666–677).

[2] Connolly, T., & Begg, C., 2005, Database Systems: A Practical Approach in Design, Implementation, and Management. Fourth Edition. Addison Wesley. Longman Inc., USA.

[3] Eyal, I., Birman, K., & Renesse, R. Van. 2015. Cache Serializability: Reducing Inconsistency in Edge Transactions. In *Proceedings - International Conference on Distributed Computing Systems*,Vol.2015–July, pp. 686–695.

[4] Kaur, K., Dhindsa, K. Singh. 2014. Hybrid Approach for Improvement of Web page

Response Time. *International Journal of Computer Science and Information Technologies*, Vol. 5 (5), 6755-6759.

[5] Liu, X., Ma, Y., Liu, Y., Xie, T., & Huang, G. 2016. Demystifying the Imperfect Client-Side Cache Performance of Mobile Web Browsing. *IEEE Transactions on Mobile Computing*, *15*(9), 2206–2220.

[6] Nanda, P., Singh, S., & Saini, G. 2015. A Review of Web Caching Techniques and Caching Algorithms for Effective and Improved Caching General Terms. *International Journal of Computer Applications*, *128*(10), 975–8887.

[7] Nikolaou, S., Van Renesse, R., & Schiper, N. 2016. Proactive Cache Placement on Cooperative Client Caches for Online Social Networks. *IEEE Transactions on Parallel and Distributed Systems*, *27*(4), 1174–1186.

[8] Mertz, J.. 2017. Understanding and Automating Application-Level Caching. Thesis, *Universidade Federal Do Rio Grande Do Sul Instituto De Informática Programa De Pós-Graduação Em Computação*.

[9] Mertz, J., & Nunes, I. 2016. Seamless and Adaptive Application-level Caching. In *VI Workshop de Teses e Dissertações do CBSoft (WTDSoft 2016)* (p. 7).

[10] Zhang, L., Floyd, S., Jacobson, V. 1997. Adaptive Web Caching. In: Proceedings of the NLANR Web Cache Workshop, Boulder, CO.