Pengujian Statis pada Sistem Server Web Berbasis *Cluster* dengan Algoritme *Never Queue*

Nongki Angsar[1]

***Abstract*—The increase of web traffic and the development of network bandwidth are relatively faster than the development of microprocessor technology. This causes one point server platform no longer adequate to meet the needs of system scalability web server. Thus, multiple server platforms are the answer. One solution that has been known is a cluster-based web server system. This paper designs and develops cluster based web server system with Never Queue Algorithm. The system is then tested with a web workload distribution testing. The testing is carried out by generating HTTP workloads statically (with fixed HTTP request rate) from the client to a web server system pool. The result shows that HTTP requests are well-distributed to web server system pool by Never Queue Algorithm. HTTP reply rate tends to be stable at average of 1,031.8 replies/s because the system is saturated, while TCP connection rate, response time, and error, tend to rise along with the rise of HTTP request rate. As for throughput, the average is 2.983 Mbps. Correlation between stability and the rise of error is, at the saturated point, bigger HTTP requests yield bigger errors.**

***Intisari*—Peningkatan lalu-lintas web dan perkembangan *bandwidth* jaringan yang relatif lebih cepat dari perkembangan teknologi mikroprosesor dewasa ini menyebabkan platform server satu titik tidak lagi memadai untuk memenuhi kebutuhan *scalability* sistem server web. Platform server jamak adalah jawabannya. Salah satu solusi yang telah dikenal adalah sistem server web berbasis *cluster*. Dalam makalah ini dilakukan rancang bangun sistem server web berbasis *cluster* dengan algoritme *Never Queue* dan dilanjutkan dengan pengujian distribusi beban kerja web pada sistem ini. Pengujian dilakukan dengan cara menghasilkan beban kerja HTTP secara statis (dengan pesat permintaan HTTP per detik yang tetap) dari klien ke *pool* sistem server web. Dalam makalah ini, hasil pengujian secara statis dengan pesat permintaan HTTP per detik yang tetap menunjukkan bahwa algoritme *Never Queue* mendistribusikan permintaan HTTP ke *pool* sistem server web dengan baik dan mendapatkan balasan HTTP yang cenderung stabil di rata-rata pesat balasan HTTP sebesar 1.031,8 *replies*/s karena sistem mencapai titik jenuh pelayanan. Sedangkan pada parameter pesat koneksi TCP, waktu tanggapan, dan galat terjadi kenaikan seiring dengan bertambahnya pesat permintaan HTTP yang dihasilkan. Adapun *throughput* rata-rata berada di angka 2,983 Mbps. Korelasi kestabilan layanan dengan kenaikan galat terjadi saat mencapai titik jenuh. Semakin besar permintaan HTTP, semakin besar galat yang terjadi.**

***Kata Kunci*—Pengujian Distribusi, Server Web, *Cluster*.**

I. PENDAHULUAN

Seiring dengan semakin kompleksnya layanan dan aplikasi web dalam berbagai bidang, permintaan layanan web dari pengguna semakin meningkat. Contoh layanan dan aplikasi web yang populer adalah layanan dan aplikasi bisnis *(e-business)*, pendidikan *(e-learning)*, berita *(e-news)*, dan lain-lain.

Demikian pula dengan perkembangan infrastruktur jaringan dan komunikasi komputer, semakin tahun semakin baik. Penerapan serat optis pada kabel [1], Gigabit Ethernet pada LAN [2], *broadband*-ISDN pada WAN [3], transmisi digital xDSL pada jalur telepon [3], dan modem kabel membuat *bandwidth* jaringan semakin besar. Bahkan sebuah prediksi pada tahun 1995 memperkirakan bahwa perkembangan *bandwidth* jaringan akan berlipat tiga kali setiap tahun untuk 25 tahun mendatang [4]. Prediksi ini masih berlaku, khusus untuk serat optis, merujuk pada tulisan yang dibuat pada tahun 2008 [5].

Di satu sisi, perkembangan komputer (jumlah transistor dalam keping mikroprosesor), menurut prediksi pada tahun 1960-an, hanya akan berlipat dua kali setiap 18 bulan [6]. Prediksi ini sudah terbukti bertahun-tahun hingga saat ini dan lazim disebut dengan hukum Moore *(Moore's Law).*

Dengan melihat fakta perkembangan *bandwidth* jaringan yang berlipat lebih dari dua kali perkembangan komputer dan melihat kompleksnya perkembangan layanan dan aplikasi web, maka kemungkinan kemacetan di masa mendatang akan terletak pada sisi server.

Makalah ini bertujuan untuk merancang dan membangun sebuah sistem yang dapat mengatasi kemacetan pada sisi server ini.

II. UPAYA MENGATASI KEMACETAN SERVER

Ada dua upaya yang dapat dilakukan untuk mengatasi kemacetan server, yaitu upaya *scale-up* (platform server tunggal) dan upaya *scale-out* (platform server jamak). Upaya pertama sudah cukup baik, tetapi mempunyai beberapa kelemahan. Pertama, upaya tersebut membutuhkan biaya yang besar agar dapat selalu mengikuti perkembangan teknologi mutakhir. Kedua, tidak dapat menghilangkan fakta bahwa titik tunggal kegagalan (*Single Point of Failure*, SPOF) justru ada pada server itu sendiri. Ketiga, keberlangsungan dan ketersediaan layanan akan terganggu oleh peningkatan *scalability* server. Keempat, penggantian ke perangkat keras baru menyebabkan perangkat keras lama cenderung tidak terpakai lagi dalam sistem. Sedangkan upaya kedua, sebaliknya, lebih murah dan tidak memiliki SPOF.

Salah satu sistem server web jamak yang populer dan banyak digunakan adalah sistem server web berbasis *cluster* [7]. Sistem ini memiliki *scalability*, availabilitas, *manageability*, dan *cost-effectivity* yang tinggi dalam melayani permintaan web dari klien dari berbagai penjuru dunia.

1. *Scalability* adalah kemampuan jaringan untuk melakukan ekspansi saat beban layanan meningkat. *Scalability* dapat

[1]*Politeknik Negeri Kupang, Jl. Adi Sucipto Penfui, Kampus Undana Baru, Kupang, NTT.,(email: angsar.nongki@gmail.com)*

dicapai dengan cara menambahkan titik-titik baru dalam *cluster*
2. Availabilitas adalah ketersediaan jaringan untuk melayani permintaan pengguna selama 24 jam sehari dan tujuh hari seminggu (24x7). Availabilitas dapat dicapai dengan cara mendeteksi kegagalan *daemon* atau titik *real-server* dan melakukan konfigurasi ulang sistem secara tepat.
3. *Manageability* adalah kemampuan sistem untuk dapat diatur dengan mudah sekalipun dalam ukuran besar.
4. *Cost-effectivity* adalah perbandingan antara kinerja dan biaya sistem. Perbandingan ini harus sebesar atau seekonomis mungkin.

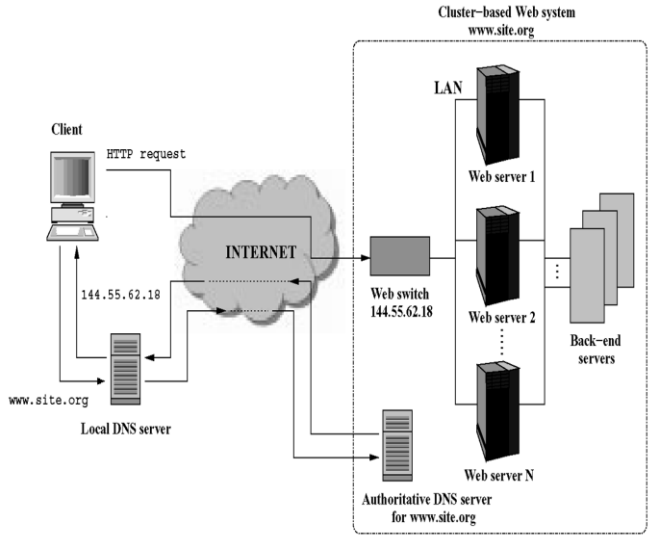
Sistem inilah yang dirancang dan dibangun serta diuji di dalam makalah ini.

III. SISTEM SERVER WEB BERBASIS *CLUSTER*

Sebuah sistem server web berbasis *cluster* adalah sekumpulan server web heterogen yang bekerja di bawah koordinasi penyeimbang beban untuk melayani permintaan HTTP dari klien. *Cluster* server web tampak dari klien sebagai satu sistem tunggal dengan satu nama dan alamat IP. Sistem ini mempunyai bagian-bagian sebagai berikut [6].

1. Penyeimbang beban adalah peranti digital yang sengaja ditempatkan pada lapis ke-7 atau ke-4 ISO/OSI untuk membagi beban kerja antar server web.
2. *Server Pool* adalah *cluster* server-server yang mengerjakan layanan sesungguhnya, seperti web, FTP, dan *mail*.
3. *Back-end Server* adalah bagian belakang sistem yang menyimpan data dan isi layanan terkait server, seperti basis data dan NFS.

Arsitektur sistem ditunjukkan dalam Gbr. 1.



Gbr. 1 Arsitektur sistem server web berbasis *cluster* [7].

Ada dua fungsi utama penyeimbang beban dalam sistem server web berbasis *cluster*, yaitu fungsi perutean (yang diwujudkan dalam mekanisme perutean) dan fungsi pengiriman (yang diwujudkan dalam algoritme pengiriman).

A. Mekanisme Perutean

Mekanisme perutean berfungsi untuk mengemas dan mengarahkan permintaan klien ke sebuah titik server web target. Mekanisme perutean yang digunakan dalam makalah ini adalah *Network Address Translation* (NAT).

B. Algoritme Pengiriman

Algoritme pengiriman berfungsi untuk memilih titik server web yang tepat dalam memberikan tanggapan atas permintaan klien [8]. Algoritme pengiriman yang digunakan dalam makalah ini adalah algoritme *Never Queue*.

C. Penentuan Bobot

Penentuan bobot dipengaruhi oleh jenis isi web (*web-content*) yang disediakan oleh server web. Apabila isi web bersifat statis (*static web-content*), bobot hanya dipengaruhi oleh faktor kecepatan media penyimpan, P_m. Apabila isi web bersifat dinamis (*dynamic web-content*), bobot hanya dipengaruhi oleh faktor kecepatan prosesor, P_p. Jika isi web merupakan gabungan statis dan dinamis, maka rumusnya menjadi (1).

$$w = \alpha P_p + (1-\alpha) P_m \quad (1)$$

dengan α adalah rasio yang menentukan besar kontribusi P_m dan P_p terhadap bobot w.

$$\alpha = \frac{N_d}{(N_d + N_s)} \quad (2)$$

dengan N_d dan N_s adalah jumlah statistik akses isi web dinamis dan statis.

IV. METODOLOGI

Metodologi yang digunakan dalam makalah ini mencakup alat dan bahan, jalannya penelitian, perancangan sistem, dan cara analisis.

A. Alat dan Bahan

Spesifikasi alat yang digunakan dalam makalah ini adalah sebagai berikut.

1. Penyeimbang beban: Intel® Celeron® Dual-Core N3060 1,6 GHz x 2, DDR3 SDRAM 2 GB, HD Toshiba® SATA 500 GB x 1, NIC Realtek PCI Fast Ethernet, Linux 4.8.6-300.
2. Real-server 1: AMD® A4-1200 APU with Radeon® HD Graphics 1GHz x 2, DDR3 SDRAM 2 GB, HD Seagate® Barracuda® ATA 500 GB x 1, NIC Realtek PCI Fast Ethernet, Windows 8 Pro, Apache 2.2.25.
3. Real-server 2: AMD® Dual Core Processor C-50 1 GHz x 2, DDR3 SDRAM 2GB, HD Hitachi® ATA 320GB x 1, NIC Atheros Family PCI, Windows 7 Ultimate, Apache 2.2.25.
4. *Klien*: Intel® Celeron® M CPU 430 1,73 GHz, DDR2 SDRAM Visipro® 512 MB, HD Seagate® Barracuda® 60 GB 5400 rpm x 1, NIC Broadcom 440x 10/100 Mbps, Linux 2.6.25-14.

5. *Switch*: SMC® 5-port 10/100Mbps Auto-MDIX Switch - SMC-EZ6505TX (*store-and-forward transmission*).
6. Kabel UTP (*Cat* 5) 15 meter.

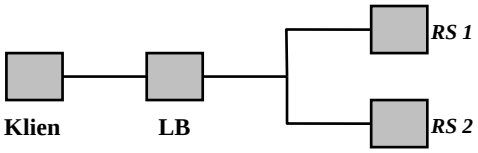
Bahan yang diteliti adalah rata-rata jumlah balasan HTTP per detik (pesat balasan HTTP) dari sistem server web berbasis *cluster* apabila jumlah permintaan HTTP per detik (pesat permintaan HTTP) oleh klien bersifat statis.

B. Jalannya Penelitian

1. Melakukan konfigurasi perangkat keras.
   a. Memotong kabel UTP *cat* 5 menjadi lima bagian dengan panjang masing-masing 3 meter per ruas.
   b. Memasang konektor RJ-45 pada masing-masing kedua ujung kabel UTP *cat* 5 dengan *crimping tool*.
   c. Menghubungkan ujung kabel UTP *cat* 5 yang sudah dipasangi konektor RJ-45 pada *port layer* 2 *switch* dan ujung kabel satunya lagi pada *port* kartu LAN komputer yang bersesuaian, masing-masing pada *port* kartu LAN komputer penyeimbang beban, *port* kartu LAN komputer *real-server* 1, *port* kartu LAN komputer *real-server* 2, dan *port* kartu LAN komputer klien.
2. Mengkonfigurasi perangkat lunak.
   a. Penyeimbang Beban
      - Konfigurasi *Kernel* dan *Compile Kernel.*
      - Konfigurasi antarmuka jaringan dan penopengan (NAT).
      - Konfigurasi perangkat lunak penyeimbang beban, mendefinisikan algoritme pengiriman, pemetaan penyeimbang beban ke *real server* 1 dan 2 (alamat dan *port*) serta penentuan bobot
   b. *Real server*
      - Konfigurasi web server *Apache* dan antarmuka jaringan pada *Real server* 1.
      - Konfigurasi web server *Apache* dan antarmuka jaringan pada *Real server* 2.
   c. DNS Server
      Konfigurasi DNS server Bind: *file named.conf, file localhost.zone, file in-addr.arpa.zone, resolv.conf*, dan *host.conf*.
   d. Klien
      - Konfigurasi antarmuka jaringan.
      - Pemasangan perangkat lunak pengujian beban kerja web.
3. Melakukan pengujian distribusi beban kerja web statis pada sistem server web berbasis *cluster*.
4. Pada akhir pengujian dilakukan pengambilan data-data berupa:
   a. Pesat permintaan HTTP (rps).
   b. Pesat balasan HTTP (*replies*/s).
   c. Waktu tanggapan (ms).
   d. *Throughput* (Mbps).
   e. Pesat koneksi TCP (cps).
   f. Galat (galat).
5. Membuat tabel berdasarkan data-data yang disebutkan pada langkah nomor 4.
6. Menyusun laporan penelitian berdasarkan hasil penelitian dan didukung literatur yang sesuai.
7. Membuat kesimpulan.

C. Perancangan Sistem

Sistem yang dirancang dalam makalah ini adalah seperti dalam Gbr. 2.



Gbr. 2 Jaringan sistem server web berbasis *cluster*.

D. Cara Analisis

Sistem server web yang dibuat kemudian divalidasi dan dievaluasi menurut tiga parameter pengujian, yaitu jumlah pesat balasan HTTP, waktu tanggapan, dan *troughput*. Ketiga parameter pengujian tersebut diuji untuk algoritme yang dipakai, yaitu *Never Queue*. Cara pengujian dilakukan dengan menghasilkan pesat permintaan HTTP (rps) dari klien secara statis, lalu mencatat jumlah pesat balasan HTTP (*replies*/s), waktu tanggapan (ms), *troughput* (Mbps), pesat koneksi TCP (cps), dan galat dari penyeimbang beban yang mengatur permintaan HTTP ke kedua *real-server*. Data-data tersebut lalu ditampilkan dalam tabel. Perbandingan keenam parameter dilakukan dengan melihat tabel data-data yang dihasilkan untuk algoritme *Never Queue*. Jadi, ada satu tabel yang berisi jumlah pesat permintaan HTTP (rps), jumlah pesat balasan HTTP (*replies*/s), waktu tanggapan (ms), *troughput* (Mbps), pesat koneksi TCP (cps), dan galat. Tabel inilah yang digunakan untuk melihat karakteristik sistem yang dibuat dengan algoritme *Never Queue* dalam mendistribusikan beban kerja web ke sistem server web berbasis *cluster*.

V. HASIL DAN PEMBAHASAN

Setelah konfigurasi perangkat keras dan konfigurasi perangkat lunak pada sistem server web berbasis *cluster* selesai, tahap selanjutnya adalah pengujian distribusi beban kerja web untuk menunjukkan karakteristik algoritme *Never Queue* dalam mendistribusikan permintaan HTTP ke kedua *real-server*. Untuk mengujinya, dibuat permintaan HTTP dari sisi klien untuk memproduksi beban secara statis dengan pesat koneksi TCP tunggal.

A. Hasil Pengujian Beban Statis

Pada pengujian ini, pesat permintaan HTTP yang dihasilkan sebesar 1.000, 1.200, 1.400, 1.600, 1.800, dan 2.000 permintaan HTTP per detik dan didistribusikan ke kedua server web dalam *cluster* (*pool*) dengan algoritme *Never Queue*. Angka-angka permintaan HTTP per detik ini diperoleh dengan metode *Trial and Error* dan akan berbeda untuk konfigurasi perangkat keras yang berbeda. Dasar penggunaan angka-angka ini karena pada angka-angka permintaan HTTP ini pesat balasan HTTP dari server sudah stabil. Tujuan pengujian distribusi beban kerja web ini adalah

untuk mengukur jumlah pesat balasan HTTP, waktu tanggapan, dan *troughput* menurut algoritme *Never Queue* di saat pesat permintaan HTTP tetap.

Untuk pengujian beban statis ini, hasilnya adalah sebagai berikut.

```
Total: connections 5000 requests 49600 replies
49600 test-duration 50.158 s

Connection rate: 99.7 conn/s (10.0 ms/conn, <=47
concurrent connections)
Connection time [ms]: min 54.1 avg 187.3 max
418.9 median 186.5 stddev 63.5
Connection time [ms]: connect 12.8
Connection length [replies/conn]: 10.000

Request rate: 988.9 req/s (1.0 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 945.1 avg 989.4 max
1013.4 stddev 19.5 (10 samples)
Reply time [ms]: response 17.5 transfer 0.0
Reply size [B]: header 241.0 content 44.0 footer
0.0 (total 285.0)
Reply status: 1xx=0 2xx=49600 3xx=0 4xx=0 5xx=0

CPU time [s]: user 4.69 system 41.90 (user 9.4%
system 83.5% total 92.9%)
Net I/O: 347.7 KB/s (2.8*10^6 bps)

Errors: total 40 client-timo 40 socket-timo 0
connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0
other 0
```

Ada enam kelompok statistik dalam hasil pengujian tersebut yang dipisahkan oleh baris kosong, yaitu sebagai berikut.

1) Hasil Total: Hasil total yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Total: connections 5000 requests 49600 replies
49600 test-duration 50.158 s
```

Bagian ini menyatakan bahwa ada 5.000 koneksi TCP yang dibuat oleh klien, 49.600 permintaan yang dikirim keluar, 49.600 balasan yang diterima, dan total waktu tes 50,158 detik.

2) Hasil Koneksi TCP: Pesat koneksi TCP dan jumlah koneksi TCP bersamaan yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Connection rate: 99.7 conn/s (10.0 ms/conn, <=47
concurrent connections)
```

Baris ini menunjukkan pesat koneksi sebesar 99,7 koneksi/detik (10,0 milidetik/koneksi) dan paling sedikit ada 47 koneksi yang dibuka secara bersamaan ke server pada suatu waktu.

Statistik waktu hidup satu koneksi TCP penuh yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Connection time [ms]: min 54.1 avg 187.3 max
418.9 median 186.5 stddev 63.5
```

Baris ini menunjukkan statistik waktu-hidup koneksi yang berhasil. Waktu-hidup koneksi adalah waktu yang terhitung sejak koneksi TCP diadakan hingga koneksi ditutup. Sebuah koneksi dinyatakan berhasil apabila memiliki paling sedikit satu permintaan yang telah dibalas oleh server. Dari baris tersebut dapat dilihat bahwa waktu-hidup minimum koneksi adalah 54,1 milidetik, rata-rata 187,3 milidetik, maksimum 418,9 milidetik, dan median 186,5 milidetik, dengan deviasi standar 63,5 milidetik.

Rata-rata waktu yang dibutuhkan untuk membentuk suatu koneksi TCP dengan server, termasuk koneksi TCP yang berhasil maupun yang gagal atau tidak mendapat balasan, yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Connection time [ms]: connect 12.8
```

Baris ini menyatakan bahwa dibutuhkan 12,8 milidetik untuk membentuk koneksi TCP dengan server.

Rata-rata balasan HTTP per koneksi TCP yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Connection length [replies/conn]: 10.000
```

Baris ini menyatakan bahwa rata-rata balasan per koneksi TCP adalah 10 (menunjukkan protokol HTTP/1.1 – *persistent connection*). Untuk protokol HTTP/1.0, rata-rata balasan per koneksi TCP adalah 1,0.

3) Hasil Permintaan HTTP: Hasil permintaan HTTP yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Request rate: 988.9 req/s (1.0 ms/req)
```

Baris ini menunjukkan pesat permintaan HTTP yang dikeluarkan oleh klien ke server dan waktu yang dibutuhkan untuk mengeluarkan satu permintaan HTTP. Dari hasil tersebut dapat dilihat bahwa pesat permintaan HTTP yang dihasilkan adalah 988,9 permintaan/detik yang berarti dibutuhkan waktu 1,0 milidetik untuk menghasilkan satu permintaan HTTP.

Rata-rata ukuran permintaan HTTP dalam satuan *byte* yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Request size [B]: 75.0
```

Baris ini menunjukkan bahwa rata-rata ukuran permintaan HTTP yang dihasilkan adalah 75 *byte*.

4) Hasil Balasan HTTP: Statistik pesat balasan HTTP yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Reply rate [replies/s]: min 945.1 avg 989.4 max
1013.4 stddev 19.5 (10 samples)
```

Baris ini menunjukkan statistik pesat balasan HTTP, yaitu minimum 945,1 balasan/detik, rata-rata 989,4 balasan/detik, dan maksimum 1.013,4 balasan/detik, dengan deviasi standar 19,5 balasan/detik.

Waktu respons dan transfer server yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Reply time [ms]: response 17.5 transfer 0.0
```

Baris ini memberikan informasi lama waktu yang dibutuhkan oleh server untuk merespons permintaan klien dan waktu yang dibutuhkan klien untuk membaca balasan server. Waktu respons dihitung sejak *byte* pertama permintaan dikirim hingga *byte* pertama balasan diterima oleh klien. Waktu baca

(transfer) adalah waktu yang dibutuhkan untuk membaca keseluruhan balasan, terutama apabila ukuran balasan cukup besar sehingga harus terfragmentasi dalam beberapa segmen TCP. Dari hasil tersebut dapat dilihat bahwa waktu respons adalah sebesar 17,5 milidetik dan waktu transfer sebesar 0 milidetik.

Ukuran kepala, isi, kaki, dan total balasan HTTP dalam satuan *byte* yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Reply size [B]: header 241.0 content 44.0 footer
0.0 (total 285.0)
```

Baris ini menunjukkan bahwa ukuran kepala balasan adalah 241 *byte*, ukuran isi balasan 44 *byte*, ukuran kaki balasan 0 *byte*, dan ukuran total balasan adalah 285 *byte*.

Status balasan HTTP yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Reply status: 1xx=0 2xx=49600 3xx=0 4xx=0 5xx=0
```

Baris ini menunjukkan bahwa ada 49.600 balasan dalam kode status 2xx, yang berarti balasan telah "sukses" dikirimkan.

5) Penggunaan CPU dan Jaringan: Waktu penggunaan CPU yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
CPU time [s]: user 4.69 system 41.90 (user 9.4%
system 83.5% total 92.9%)
```

Baris ini menunjukkan bahwa dibutuhkan waktu 4,69 detik (9,4%) dalam mode "user" dan 41,90 detik (83,5%) dalam mode "system" untuk mengeksekusi program. Idealnya jumlah dalam kedua mode adalah 100%. Apabila persentase penggunaan CPU secara signifikan kurang dari 100%, berarti ada program lain yang sedang dijalankan saat program dieksekusi, sehingga pengujian harus diulang.

Nilai *throughput* jaringan yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Net I/O: 347.7 KB/s (2.8*10^6 bps)
```

Nilai *throughput* jaringan dihitung dari jumlah *byte* yang dikirim dan diterima dalam koneksi TCP. Ini berarti hanya bagian *payload* protokol saja yang dihitung (tidak termasuk *header* protokol), tanpa memperhitungkan transmisi-ulang yang mungkin terjadi pada tingkat TCP. Baris tersebut menunjukkan bahwa nilai *throughput* jaringan adalah sebesar 347,7 KBps atau 347,7 x 1.024 x 8 = 2.848.358,4 bps ($2{,}8 \times 10^6$ bps = 2,8 Mbps).

6) Galat: Statistik galat yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Errors: total 40 client-timo 40 socket-timo 0
connrefused 0 connreset 0
```

Baris ini menunjukkan bahwa secara total terjadi 40 galat. Ada galat yang disebabkan oleh terlewatinya batas waktu (*client-timo* = 40), tidak terjadi kegagalan koneksi TCP akibat terlewatinya batas waktu pada tingkat *socket* (*socket-timo* = 0), tidak terjadi kegagalan koneksi TCP akibat penolakan server, dan tidak ada kegagalan koneksi TCP akibat penutupan oleh server.

Galat lainnya yang muncul dalam baris mode teks Linux tampak sebagai berikut.

```
Errors: fd-unavail 0 addrunavail 0 ftab-full 0
other 0
```

Baris ini menunjukkan bahwa klien tidak pernah memproduksi beban melebihi batas yang ada dalam *file-descriptor*, klien selalu mendapat nomor *port*, tabel *file-descriptor* tidak pernah penuh, dan tidak terjadi jenis galat lainnya.

Setelah pengujian pertama, selanjutnya diadakan pengujian kedua hingga keenam. Hasilnya diperlihatkan dalam Tabel I.

TABEL I
HASIL PENGUJIAN STATIS ALGORITME *NEVER QUEUE*

Pesat Permintaan HTTP (rps)	Pesat Balasan HTTP (*replies*/s)	Waktu Tanggapan (ms)	*Through-put* (Mbps)	Pesat Koneksi TCP (cps)	Galat (galat)
1.000	989,4	17,5	2,8	99,7	40
1.200	1.058,3	140,4	3,1	114,5	358
1.400	1.036,6	191,3	3,0	132,5	1.063
1.600	1.015,3	204,0	3,0	151,3	1.604
1.800	1.041,9	202,7	3,0	167,9	1.870
2.000	1.049,4	203,5	3,0	187,4	2.182

Tabel I menunjukkan, saat mencapai titik jenuh, semakin besar permintaan HTTP, semakin besar galat yang terjadi.

VI. KESIMPULAN

Kesimpulan yang dapat diambil dari pengujian ini adalah sebagai berikut. Dalam pengujian beban statis dengan pesat permintaan HTTP 1.000 rps, 1.200 rps, 1.400 rps, 1.600 rps, 1.800 rps, dan 2.000 rps, jika dilihat dalam Tabel I, maka parameter pesat balasan HTTP cenderung stabil di rata-rata pesat balasan HTTP sebesar 1.031,8 *replies*/s karena sistem mencapai titik jenuh layanan. Sedangkan untuk parameter pesat koneksi TCP, pada waktu tanggapan dan galat terjadi kenaikan seiring dengan bertambahnya pesat permintaan HTTP yang dihasilkan. Adapun *throughput* rata-rata berada di angka 2,983 Mbps. Korelasi kestabilan layanan dengan kenaikan galat adalah saat mencapai titik jenuh, semakin besar permintaan HTTP, semakin besar galat yang terjadi.

REFERENSI

[1] Roger L. Freeman, *Telecommunication Transmission Handbook*, 4th edition, Canada: John Wiley & Sons, Inc., 1998.

[2] H. Kaplan dan B. Noseworthy, *The Ethernet Evolution: 10 to 10,000 Mbps*, Atlanta: Networld Interop, 2000.

[3] William Stallings, *Data and Computer Communication*, 6th edition, Upper Saddle River, New Jersey: Prentice-Hall, 2000.

[4] J. Gray dan P. Shenoy, "Rules of Thumb in Data Engineering", *Proc. of 16th International Conference on Data Engineering*, 2000, hal. 1-8.

[5] G. Gilder (2008) The Coming Creativity Boom, [Online], http://www.forbes.com/forbes/2008/1110/036.html, tanggal akses: 3 Agustus 2018.

[6] *IA-32 Intel® Architecture Software Developer's Manual Vol. 1: Basic Architecture, Order Number 24547-012*. Intel Corporation, 2003.

[7] V. Cardellini, E. Casalicchio, M. Colajanni, dan P.S. Yu, "The State of the Art in Locally Distributed Web-server Systems," IBM Research Report, 2001.

[8] N. G. Shivaratri, P. Krueger, M. Singhal, "Load Distributing for Locally Distributed Systems," *Computer*, Vol. 25, No. 12, Dec. 1992.