BAB II
DASAR TEORI

2.1 Jaringan Komputer

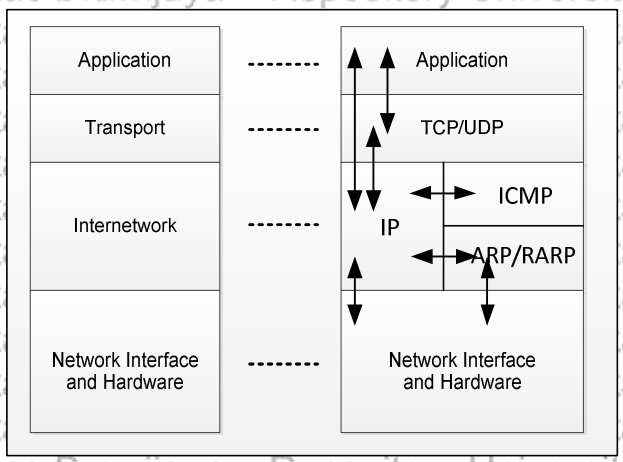
Jaringan Komputer adalah sekelompok komputer otonom yang saling berhubungan antara satu dengan lainnya menggunakan protokol komunikasi melalui media komunikasi sehingga dapat saling berbagi informasi, program-program, penggunaan bersama perangkat keras seperti *printer, harddisk*, dan sebagainya. Selain itu jaringan komputer bisa diartikan sebagai kumpulan sejumlah terminal komunikasi yang berada di berbagai lokasi yang terdiri dari lebih satu komputer yang saling berhubungan [3].

2.2 Protokol TCP/IP (*Transmission Control Protocol/Internet Protocol*)

TCP/IP adalah serangkaian protokol yang memungkinkan komunikasi antara beberapa komputer. Dulu TCP/IP ini tidak begitu penting bagi komputer-komputer untuk saling berkomunikasi karena bukan protokol yang umum. Tapi saat komputer saling terkoneksi dalam jaringan, kebutuhan itu muncul saat komputer-komputer tersebut sepakat untuk menggunakan protokol-protokol tertentu.

Seperti pada perangkat lunak, TCP/IP dibentuk dalam beberapa lapisan (*layer*). Dengan dibentuk dalam *layer*, akan mempermudah dalam pengembangan dan pengimplementasian. Antar *layer* dapat berkomunikasi ke atas maupun ke bawah dengan suatu penghubung interface. Tiap-tiap *layer* memiliki fungsi dan kegunaan yang berbeda dan saling mendukung *layer* diatasnya. [4]

Pada protokol TCP/IP dibagi menjadi 4 *layer*, tampak pada gambar 1 :



Gambar 2.1 : Protokol TCP/IP
Sumber : Dhoto (2007:3)

Layer/lapisan aplikasi digunakan pada program untuk berkomunikasi menggunakan TCP/IP, misalnya *Telnet* dan *File Transfer Protocol* (FTP). Lapisan *transport* memberikan fungsi pengiriman data secara *end-to-end* ke sisi *remote* sehingga aplikasi yang beragam dapat melakukan komunikasi secara serentak, misalnya pada TCP atau UDP. Lapisan *internetwork* atau *layer network* memberikan "*vitual network*" pada *internet* seperti pada *internet protocol* (*IP*) yang memberikan fungsi *routing* pada jaringan dalam pengiriman data. Lapisan *network interface* disebut juga lapisan *link* atau *datalink layer*, yang merupakan perangkat keras pada jaringan. Contoh : IEEE802.2, X.25, ATM, FDDI, dan SNA [4].

2.3 Topologi Jaringan

Topologi pada dasarnya adalah peta dari sebuah jaringan. Topologi jaringan terbagi lagi menjadi dua, yaitu topologi secara fisik (*physical topology*) dan topologi secara logika (*logical topology*). Topologi secara fisik menjelaskan bagaimana susunan dari kabel dan komputer dan lokasi dari semua komponen jaringan. Sedangkan topologi secara logika menetapkan bagaimana informasi atau aliran data dalam jaringan [4].

Kabel atau koneksi dalam *physical topology* seringkali mengenai media jaringan (atau media fisik). Memilih bagaimana komputer-komputer akan dihubungkan dalam suatu jaringan sangat penting (terlebih lagi dalam jaringan perusahaan). Beberapa topologi yang sering digunakan antara lain adalah *star*, *bus*, *mesh*, dan *ring*. Dalam topologi *star*, semua kabel dihubungkan dari komputer-komputer ke lokasi pusat (*central location*), dimana semuanya terhubung ke suatu alat yang dinamakan *hub/switch* [3].

2.4 Sistem Operasi

Sistem operasi merupakan sistem perangkat lunak komputer yang menyediakan kemudahan dan cara yang efisien bagi pengguna untuk menjalankan program-program. Sistem operasi mengatur urutan suatu proses untuk dijalankan, menyipan dan menerima data secara efisien, menjalankan program yang mungkin digunakan secara bersamaan, serta mengaatur penggunaan *resource* secara bersamaan.

Pelayanan sistem operasi dapat dilakukan dalam kegiatan-kegiatan berikut :

- *CPU Scheduling,* mengatur waktu penggunaan CPU diantara proses-proses

- *Memory management,* mengatur penggunaan memori oleh proses-proses
- *Swapping*, Memindahkan proses dan data-data dari memori ke media penyimpanan dan sebaliknya
- Pendukung perangkat *input/output,* menyediakan program pengendali perangkat keras (*driver)* untuk mendukung keperluan perangkat *input/output*.
- Sistem berkas, mengatur penyimpanan data/ *code* dalam media penyimpanan ke dalam *file* dan *directory*.
- *Command interface*, secara tekstual atau grafis, untuk mengamati dan memanipulasi proses.
- *System Calls,* memungkinkan dalam mengakses fasilitas dari sistem operasi.
- Proteksi, mennjaga agar proses-proses tidak mengganggu satu sama lain, meliputi penggunaan *resource* dan data.
- Komunikasi, menyediakan sarana komunikasi bagi pengguna dan program-program di dalam suatu komputer maupun antar komputer dalam jaringan.

Proses dalam sistem operasi dapat didefinisikan sebagai suatu program yang dijalankan, suatu entity yang ditujukan untuk dijalankan dalam prosesor, atau suatu unit aktifitas uang bersifat sebagai *thread* sekuensial tunggal, keadaan kini *(current state),* himpunan dari *resource.* Proses memerlukan memory sebagai *resource* yang akan diisi instruksi, data, *stack,* dan *heap* [5].

2.4.1 Memori

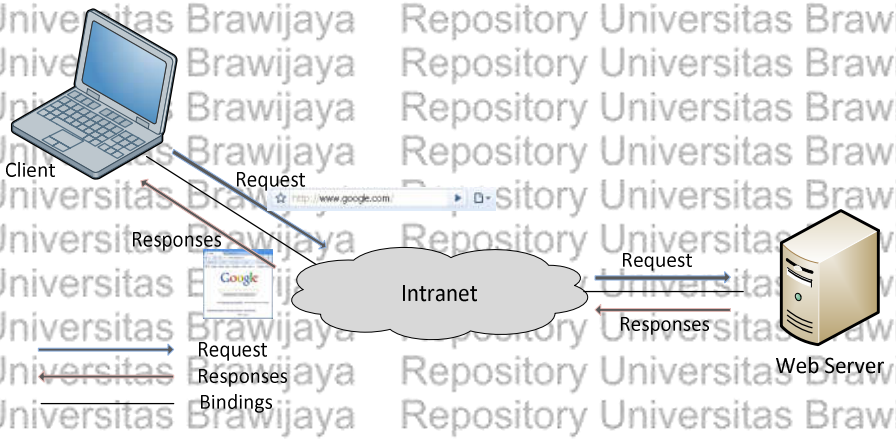
Memori merupakan salah satu sumber daya yang penting dalam pengeksekusian sebuah proses. Agar suatu proses dapat dieksekusi, ia harus terletak di dalam memori sebelum CPU mengambil instruksi-instruksi pada alamat yang ditunjuk oleh program *counter*. Pemetaan memori memetakan alamat logis yang dihasilkan CPU ke alamat fisik yang nantinya akan dibawa ke memori pada saat akan dieksekusi.Pada pemetaan memori ini terdapat limit *register* yang terdiri dari rentang nilai alamat logis (*range of logical address*) [5].

Pagging atau berbagi data pada memori memungkinkan beberapa proses untuk mengakses kode yang sama namun dengan data yang berbeda-beda. Hal ini jelas akan mengurangi jumlah ruang memori yang dibutuhkan untuk memenuhi permintaan beberapa proses tersebut [5].

2.5 *Hypertext Transfer Protocol* (HTTP)

Hypertext transfer protocol (HTTP) merupakan protokol yang digunakan untuk jenis layanan *world wide web* (WWW) pada jaringan TCP/IP. Pengembangan HTTP dikoordinasi oleh konsorsium WWW dan IETF (*internet engineering task force*) dan dipublikasikan melalui kumpulan RFC (*request for comments*). RCF 2616 mendefinisikan HTTP/1.1 yang merupakan versi HTTP yang saat ini umum digunakan [6].

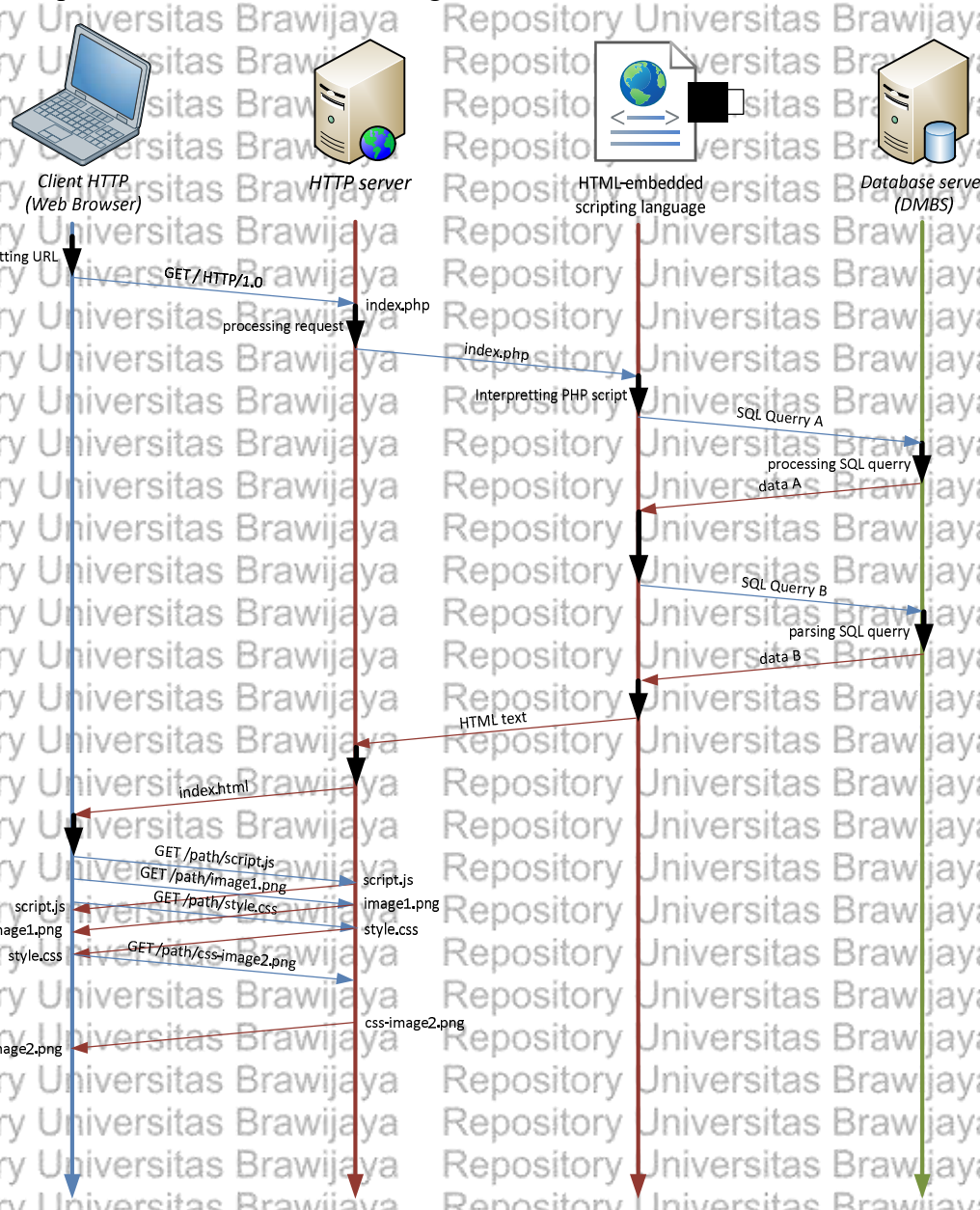
Sebuah HTTP *client* memulai *request* dengan membuat koneksi TCP (*Transmission Control Protocol*) menuju *server* (umumnya adalah *port* 80). Sedangkan HTTP *server* menunggu adanya pesan *request* pada *port* yang telah ditentukan. Setelah menerima *request* dari *client*, *server* kemudian mengirimkan *status line* antara lain "HTTP/1.1 200 OK". Setelah itu dilanjutkan dengan mengirimkan *file* yang diinginkan *client* beserta pesan kesalahan atau informasi lainnya. HTTP diidentifikasi menggunakan *uniform resource identifier* (URI) dengan format penulisan tertentu. [6]



Gambar 2.2 *Web Server*
Sumber : Ricart (1996:36)

Protokol HTTP bersifat *request-response*, yaitu dalam protokol ini *client* menyampaikan pesan *request* ke *web server*, dan *web server* kemudian memberikan *response* yang sesuai dengan *request* tersebut. *Request* dan *response* dalam protokol HTTP disebut sebagai *request chain* dan *response chain*. Hubungan HTTP yang paling sederhana adalah terdiri atas hubungan langsung antara *user agent* dengan *server* asal seperti pada gambar 2.2.

Dalam pengujian *client* melakukan permintaan berisi "GET / HTTP/1.0" yang akan diterjemahkan *web server* sebagai sebuah permintaan file *index.php* pada *directory root httpd*. *Web server* akan melakukan interpretasikan *file index.php* dan mengirimkan *file* hasil interpretasi dalam format *html* dengan nama *index.html*.



Gambar 2.3 Proses permintaan halaman web
Sumber : Ricart (1996:37) [7]

Sebuah *file index.html* yang sampai pada *client* akan diterjemahkan *web browser* untuk ditampilkan pada layar monitor dalam format yang mudah dibaca oleh pengguna. Suatu halaman *website* umumnya juga memerlukan gambar, *client-script* seperti *javascript (js)* dan *css*. *File* gambar dan *script* tersebut akan diminta kembali dalam suatu permintaan (*request)* baru ke *web server*. Permintaan bisa terjadi beberapa kali

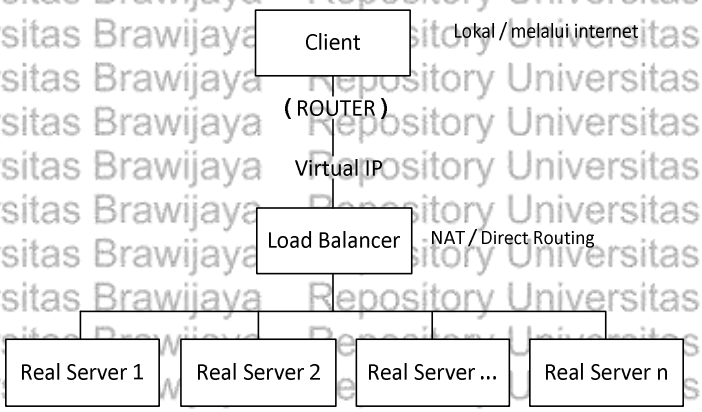
sampai satu halaman utuh selesai diunduh [7].

2.6 Linux *Virtual Server* (LVS)

Linux adalah sistem operasi yang bersifat *multiuser*, *multitasking* yang berbasis UNIX dan bersifat *freeware*, yaitu *software* yang bebas didistribusikan dan dikembangkan oleh semua orang [8].

Seperti halnya UNIX, sistem operasi Linux bergantung pada apa yang disebut dengan *kernel* yang berisi seluruh informasi *hardware* dan *interface* yang ada pada PC. *Kernel* Linux pada awalnya dikembangkan untuk CPU berbasis 80386. 80386 pada awalnya didesain untuk *multitasking*, namun sistem operasi yang masih ada bersifat *single-task*, yaitu MS-DOS. Untuk itu Linux memanfaatkan seluruh fasilitas yang ada pada 80386, terutama manajemen memori dan *floating emulation* yang membuat Linux bisa berfungsi pada prosesor yang tidak memiliki *co-prosesor* atau prosesor pengolah data matematis [8].

Linux *Virtual Server* atau disingkat LVS merupakan suatu teknologi *clustering* yang dapat digunakan untuk membangun suatu *server* dengan menggunakan kumpulan dari beberapa buah *real server*. LVS merupakan implementasi dari komputer *cluster* dengan metode *high availbalility*. LVS mengimbangi berbagai bentuk dari *service* jaringan pada banyak mesin dengan memanipulasi paket sebagaimana diproses TCP/IP *stack*. Satu dari banyak peran yang paling umum dari *Linux Virtual Server* adalah bertindak sebagai *server* yang berada pada garis terdepan dari kelompok *web server*. Seperti ditunjukkan pada gambar 2.4, linux *virtual server* atau LVS ini terdiri dari sebuah *load balancer* dan beberapa *real server* yang bekerja bersama dan memberikan layanan terhadap permintaan *user*. Permintaan *user* diterima oleh *load balancer* yang seolah-olah berfungsi sebagai IP *router* yang akan meneruskan paket permintaan *user* tersebut pada *real server* yang siap memberikan layanan yang diminta. **[2]**



Gambar 2.4 : Struktur dasar Linux *virtual server* dengan n buah *real server*
Sumber : Alex (2009:56)

Dengan demikian *virtual server* akan terdiri dari beberapa komputer yang mempunyai *image* yang sama tetapi ditempatkan pada IP yang berbeda. *User* dapat mengakses *virtual server* tersebut dengan bantuan suatu *load balancer*, yang bertugas untuk melakukan pemetaan IP dari *server* dan komputer lainnya yang berperan sebagai *virtual server*.

- LVS adalah modifikasi dari sistem Linux yang bertanggung jawab untuk mendistribusikan permintaan *client* terhadap *real server* pada kelompok *server*. *Real server* melakukan pekerjaan untuk memenuhi permintaan serta memberikan atau membuat laporan balik kepada *client*.
- LVS memelihara rekaman daripada sejumlah permintaan yang telah ditangani oleh masing-masing *real server* dan menggunakan informasi ini ketika memutuskan *server* mana yang akan ditugaskan untuk menangani suatu permintaan berikutnya.
- LVS juga dapat memiliki *load balancer* cadangan yang akan menggantikan bilamana suatu saat *load balancer* utama mengalami suatu kegagalan sehingga membentuk suatu LVS *failover*.
- Masing-masing *real server* dapat bekerja dengan menggunakan berbagai sistem operasi dan aplikasi yang mendukung TCP/IP dan *ethernet*. Pembatasan dan pemilihan sistem operasi pada *real server* serta jenis servis yang didukung oleh *realServer* dilakukan pada saat proses konfigurasi LVS dijalankan.

Layanan yang dapat didukung oleh LVS, antara lain:

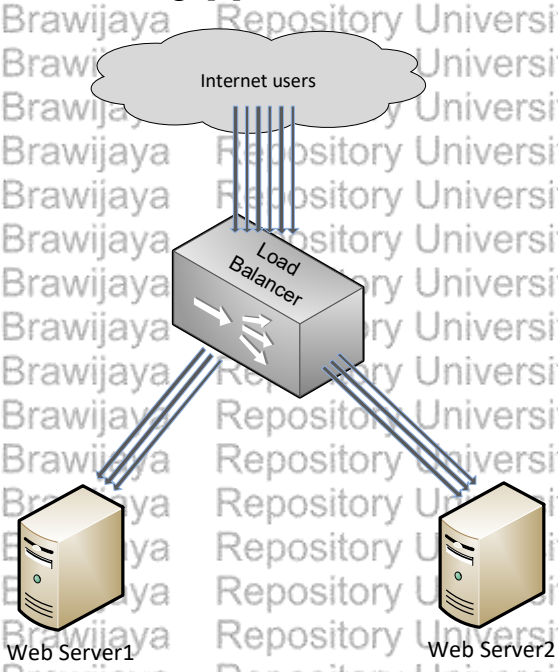
a. *Web*site (HTTP, HTTPS)
b. FTP (*File Transfer Protocol*)
c. *Email* (SMTP, POP3, dan IMAP)
d. DNS (*Domain Name Service*)
e. LDAP (*Lightweight Directory Access Protocol*)

2.7 *Load balancing*

Load balancing atau penyeimbangan beban dalam jaringan sangat penting bila skala dalam jaringan komputer makin besar atau lalu lintas data yang ada dalam jaringan komputer semakin meningkat. Layanan *load balancing* memungkinkan pengaksesan sumber daya dalam jaringan didistribusikan ke beberapa *server* lainnya supaya tidak terpusat sehingga kerja sistem *server* secara keseluruhan bisa lebih stabil [2].

Ketika sebuah sebuah *server* sedang diakses oleh *client*, maka sebenarnya *server* tersebut sebenarnya sedang terbebani karena harus melakukan proses permintaan kepada *client*. Jika *client* banyak maka proses di dalam *server* juga semakin banyak. Sesi komunikasi dibuka oleh *server* sehingga memungkinkan *client* menerima layanan dari *server* tersebut. Jika menggunakan satu *server* saja, tentu *server* tersebut suatu saat akan mengalami kejenuhan dalam melayani permintaan dari *client* karena kemampuan perangkat keras dalam melakukan proses ada batasnya.

Salah satu solusi ideal adalah dengan membagi beban yang datang ke beberapa *server* sehingga permintaan dari *client* tidak hanya terpusat pada satu perangkat keras saja. Teknik ini disebut *load balancing*. [2]



Gambar 2.5 *Load Balancing pada web server*
Sumber : linuxvirtualserver.org [9]

Dalam *load balancing*, sebuah *load balancer* (pembagi beban) bertindak sebagai *server* dengan menambahkan IP alias ke sebuah *network interface card* sehingga IP tersebut berfungsi sebagai *virtual server*. Sementara *server* lain meminjam IP tersebut dengan menambahkannya sebagai IP alias pada *loopback-interface*. Jika ada sebuah paket masuk yang ditujukan ke alamat *virtual* tadi, *load balancer* akan meneruskan ke *server* lain yang telah memiliki alamat *virtual* pada *loopback interface*.

Direct routing adalah salah satu metode dalam meneruskan paket datang dari *client* ke *real server* pada *virtual server*, selain itu juga bisa dengan menggunakan metode

NAT (*network address translation*). Dengan menggunakan *direct routing*, paket dapat diteruskan ke *load balancer* itu sendiri. Salah satu algoritma yaitu *round robin* merupakan algoritma sederhana dan banyak digunakan oleh perangkat *load balancing* [2].

2.8 Algoritma *Round Robin*

Penjadwalan algoritma *round robin* memberikan tugas-tugas ke semua bagian dalam bentuk matematika, *round robin* memilih *server* dengan $i=(i+1) \bmod n$ setiap kali, dimana n adalah jumlah *server*. Prosedur formal penjadwalan *round robin* sebagai berikut:

```
S={S0,S1,...,S n-1},dimana n > 0;
i adalah index server yang terakhir dipilih, dimulai dari -1 ;
j=i;
do{j=(j+1) mod n;
if(Available(Sj)){
i=j;
return Si;
}
while(j!=i);
return NULL; [9]
```

2.9 Daya Listrik AC

Pada arus AC (bolak-balik) terdapat tiga jenis daya, yaitu daya yang terlihat, daya nyata yang digunakan peralatan, dan daya reaktif yang ditimbulkan oleh komponen yang bersifat kapasitif maupun induktif , secara matematis daya yang terlihat merupakan hasil penjumlahan daya nyata dengan daya reaktif secara vektor (2-1).

$$S = P + jQ \quad (2\text{-}1)$$

dengan :

P = daya nyata (watt)

Q = Daya reaktif (VAR)

S = daya yang terlihat (VA)

Dalam pengujian, daya semu merupakan hasil perkalian langsung antara tegangan rata-rata dengan arus rata-rata. Nilai rata-rata pada sumber AC sinusoida ($y=a \sin(2\pi ft)$) adalah $\sqrt{2}$ dari nilai maksimal (2-2).

$$S = V_{rms} \times I_{rms} \quad (2\text{-}2)$$

dengan :

V_{rms} = Tegangan rata-rata (volt)

I_{rms} = Arus rata-rata (ampere)

S = daya yang terlihat (VA)

Karena penjumlahan daya aktif dengan daya reaktif secara vektor maka besarnya perbandingan antara daya aktif terhadap daya semu merupakan fungsi cosinus. $\cos\varphi$= P/S Dimana $\cos\varphi$, menunjukan faktor daya sistem daya listrik bolak-balik (2-3).

$$P = S\cos\varphi \quad (2\text{-}3)$$

dengan :

I_{rms} = Arus rata-rata (ampere)

S = daya yang terlihat (VA)

P = daya nyata (watt)

$\cos\varphi$ = Faktor daya (P/S)

2.10 *Wattmeter*

Penggunaan energi listrik dapat diamati langsung menggunakan sebuah *wattmeter*. *Wattmeter digital* akan menampilkan daya nyata berdasarkan nilai tegangan, arus, dan $\cos\varphi$ yang diperoleh dari perbedaan fasa tegangan dan arus direkamnya.

2.11 Parameter Performansi *Web Server*

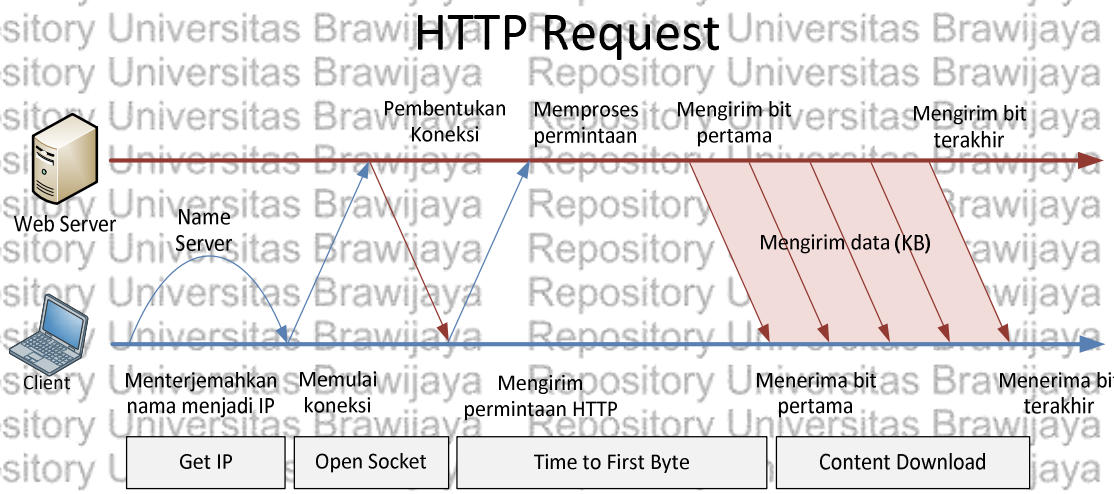
Parameter performansi pada *web server* dapat diketahui dengan melakukan pengujian :

- Jumlah *client* maksimal
- *Troughput*
- *Time to first bit* (Waktu Respon)
- *Error (Request loss)* [10]

2.11.1 Jumlah *Client* Maksimal

Jumlah *client* maksimal dapat diperoleh berdasarkan pengujian pada saat pengujian satu *server* dimana kemampuan *server* dilihat dari besar memori dan

kecepatan prosesor. Urutan proses permintaan halaman *website* pada gamabr 2.6 :



Gambar 2.6 : Permintaan HTTP
Sumber : archive.p2pu.org

Beberapa langkah yang perlu dibutuhkan untuk mendapatkan jumlah *client* maksimal yang dapat dilayani *server* adalah :

- Menentukan jumlah client maksimal yang dapat dilayani *server* (*maxClients)*.

  Sebuah *request* yang masuk ke dalam *server* akan dilayani oleh sebuah anak proses *service web server.* Dengan mengetahui jumlah memori yang digunakan setiap anak proses, dapat ditentukan berapa jumlah *request* yang dapat dilayani dengan melihat sisa memori yang tersedia dan bisa digunakan oleh *service web server*.

$$max = \frac{free}{use} \quad (2\text{-}4)$$

  dengan :

  max = Jumlah *request* maksimal yang bisa dilayani *server*.
  Free = Sisa *free memory* yang tersedia pada server.
  Use = Kebutuhan memori untuk satu proses web server.

- Menentukan waktu rata-rata yang dibutuhkan *server* dalam menangani sebuah permintaan. Melakukan beberapa kali permintaan pada saat *web server* sedang dalam keadaan tidak menerima permintaan dari *client* lain. Dengan melakukan pengujian pada *client* dapat diperoleh rata-rata waktu respon / *time to first bit* setiap *file* yang diterima *client,* berdasakan waktu respon pada *client,* dapat diperoleh waktu rata-rata yang dibutuhkan *server* dalam menangani sebuah *request.*

$$T_s = t_r - t_t \quad (2.5)$$

dengan :

T_s = Waktu proses yang terjadi pada server.
t_r = Waktu respon.
t_t = Delay pada jaringan.

- Menentukan jumlah *client* yang dapat ditangani dalam satu detik. Berdasarkan data jumlah *client* maksimal yang dapat ditangani *server* bersamaan (*maxClient)* dan waktu rata-rata yang dibutuhkan *server* dalam menangani permintaan dapat diperoleh jumlah *client* yang dapat dilayani dalam satu detik.

$$req/s = \frac{maxClients}{T_s} \quad (2\text{-}6)$$

dengan :

req/s = Permintaan yang dapat dilayani dalam satu detik.
maxClients = Permintaan yang dapat dilayani secara bersamaan.
T_s = Waktu proses yang terjadi pada server.

2.11.2 *Throughput*

Throughput definisikan sebagai rata-rata paket data yang diterima dengan benar yang dapat ditransmisikan pada satu waktu yng sama. *Throughput* pada pengujian *web server* dapat merupakan *request* yang berhasil dilayani per detik . Dengan mengetahui banyaknya *request* yang berhasil dilayani dengan benar, dapat ketahui kejenuhan *web server* sehingga *web server* dapat dikatakan *overload*. Dalam keadaan ini, permintaan tidak direspon, dan *web browser* pada *client* akan menampilkan sebuah halaman *error*.

2.11.3 *Time to First Byte* / Waktu Respon

Time to first byte merupakan selisih waktu pada saat *client* mengirimkan sebuah *request* ke *server* dan pada saat *client* menerima *byte* pertama hasil dari *request* yang dikirimkan. Permintaan sebuah *file* PHP akan membutuhkan waktu yang lebih lama pada *server* karena dalam sebuah *file* PHP terdapat *script* yang perlu diterjemahkan oleh PHP *Interpreter* atau *query database* yang memerlukan waktu proses, dibandingkan

dengan sebuah *request* file gambar yang akan langsung dilayani oleh *web server* [10].

2.11.4 *Error / Request Loss*

Request loss adalah jumlah permintaan yang tidak berhasil mendapat respon dari *server* dibandingkan dengan jumlah permintaan yang mendapat respon dengan benar. *Request* yang hilang bisa terjadi karena *packet loss,* kesalahan pada *load balancer* atau terjadi kejenuhan pada *server* [10].

Prosentase *error* ditentukan dengan Persamaan 1:

$$error\,(\%) = \frac{N_{error}}{N} x100\,\% \qquad (2\text{-}4)$$

dengan :

N_{error} = jumlah permintaan gagal
N = jumlah respon yang dikirimkan

2.12 Pengujian Kerja *Web Server* (*Stress Testing*)

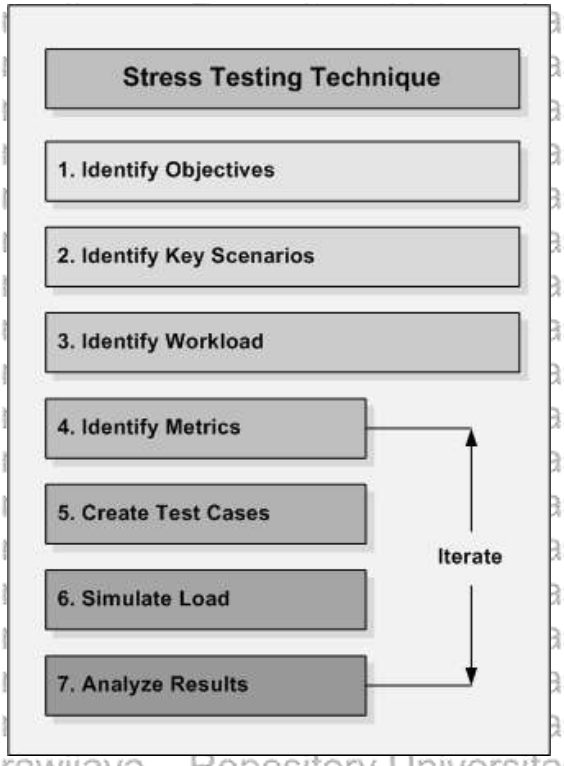
Pengujian kerja *web server* adalah sebuah pengujian yang dilakukan untuk mengetahui respon, troughput, keandalan, dan skalabilitas sebuah sistem *web server* dengan memberikan beban kerja tertentu. Pengujian umumnya dilakukan untuk :

- Menilai kesiapan produksi
- Evaluasi terhadap kriteria kinerja
- Membandingkan karakteristik kinerja beberapa sistem atau konfigurasi sistem
- Menemukan sumber dalam sebuah permasalahan
- Menentukan sebuah angka dalam melakukan *tunning* sistem
- Mencari tingkatan *troughput*

Objek yang seringkali diuji untuk penentuan nilai performansi standart dalam siklus pembangunan sebuah aplikasi *web server* antara lain

- Waktu Respon, misalnya halaman *catalog* harus dapat ditampilkan kepada *client* dalam waktu kurang dari 3 detik.
- *Troughput,* misalnya sistem harus mampu melayani 100 transaksi setiap detiknya.

- *Resource utilization,* dalam ini berarti penggunaan CPU, memori dan perangkat I/O (*network interface*).
- Jumlah *user* maksimal, menentukan berapa jumlah *user* yang dapat dilayani pada konfigurasi perangkat keras tertentu.
- Parameter yang berhubungan dengan bisnis, menghubungkan sisi bisnis yang terjadi pada saat normal atau sibuk, misalnya berapa jumlah pesanan, berapa jumlah panggilan telepon yang diterima *helpdesk* [10].



Gambar 2. 7 : Langkah melakukan *stress test*
Sumber : Meier (2007:215) [10]

Apache JMeter adalah suatu produk dari Apache yang bisa digunakan sebagai alat tes pada *server* untuk keperluan analisis dan pengukuran performa kerja dari beberapa layanan *server* yang difokuskan pada aplikasi *web*. JMeter bekerja dengan cara menggantikan fungsi dari *web browser*, yaitu mingirimkan paket *request* HTTP dalam format HTTP kemudian mengirimkan paket tersebut kepada *web server*, respon yang diberikan *server* kemudian direkam Jmeter untuk dianalisis tanpa dilakukan. Bagi *server* sendiri request yang diterima dari JMeter dianggap sama. Hasil pengujian menggunakan JMeter berupa waktu respon, *trouhput,* dan *error* dapat diperoleh menggunakan modul *listenner* yang dimiliki JMeter.

2.13 Regresi Linier Sederhana

Analisis regresi linier sederhana digunakan untuk mengetahui pengaruh dari variabel bebas terhadap variabel terikat atau dengan kata lain untuk mengetahui seberapa jauh perubahan variabel bebas dalam mempengaruhi variabel terikat. Dalam analisis regresi sederhana, pengaruh satu variabel bebas terhadap variabel terikat dapat dibuat persamaan sebagai berikut :

$$Y = a + b \cdot x \tag{2.5}$$

Dimana : Y : Variabel terikat

X : Variabel bebas

a : Konstanta

b : Koefisien Regresi.

Untuk mencari persamaan garis regresi dapat dilakukan berbagai pendekatan (rumus), sehingga nilai konstanta (a) dan nilai koefisien regresi (b) dapat dicari dengan metode sebagai berikut [11] :

$$b = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \tag{2.6}$$

$$a = \frac{\sum_{i=1}^{n} y_i}{n} - b\frac{\sum_{i=1}^{n} x_i}{n} \tag{2.7}$$

Analisis Korelasi (r) digunakan untuk mengukur tinggi redahnya derajat hubungan antar variabel yang diteliti dengan melihat koefisien korelasinya. Dengan nilai koefisien korelasi antara $-1 \leq r \leq +1$. Untuk koefisien korelasi sama dengan – 1 atau + 1 berarti hubungan kedua variabel adalah sangat erat atau sangat sempurna dan hal ini sangat jarang terjadi dalam data riil. [11]