Perbandingan Kinerja Concurrent Connection Pada Apache Http Server dan Node.js

Muhammad Fakhri Ali As Shuffi[1], Agus Prihanto[2]

[1,2] Jurusan Teknik Informatika Fakultas Teknik Universitas Negeri Surabaya
[1]muhammadfakhri.19098@mhs.unesa.ac.id
[2]agusprihanto@unesa.ac.id

***Abstrak*— Permintaan pengguna terhadap internet semakin mengalami peningkatan dari waktu ke waktu sehingga pemilihan teknologi web server yang tepat sangat penting dalam menangani permintaan pengguna yang kian meningkat. Saat ini teknologi web server yang digunakan masih didominasi oleh Apache HTTP Server. Namun, baru-baru ini Node.js mulai digandrungi pengembang web sebagai teknologi web server alternatif terbaik selain Apache dalam pengembangan web, tetapi hal ini tidak membuat popularitas Apache HTTP Server menjadi menurun. Penelitian ini bertujuan untuk mengetahui kinerja concurrent request berupa konsumsi sumber daya dan output request pada kedua web server dengan diujikan pada skenario pengujian yang paling ringan sampai ke yang paling berat. Pengujian dilakukan diatas virtual machine melalui platform Google Cloud Compute Engine. Adapun model pengujian yang dilakukan yaitu Static I/O Test, Computation Test, dan Serving File Test dengan jumlah client yang diujikan sebesar 100, 500, dan 1000 concurrent request. Indikator kinerja yang diukur diantaranya CPU, memory, total request dan error timeout. Hasil pengujian menunjukkan bahwa secara keseluruhan Node.js unggul dalam semua skenario concurrent request dan model pengujian dibandingkan dengan Apache yang mulai mengalami struggle pada 500 dan 1000 concurrent request. Node.js tercatat hanya mengalami error timeout pada pengujian Serving File Test (1000) dengan total rata-rata sebesar 5.8 error, sedangkan Apache menunjukkan kinerja terbaiknya pada skenario 100 concurrent request namun masih tidak mampu melebihi total request yang sudah diperoleh dari Node.js.**

***Kata Kunci*— Apache, Node.js, Concurrent request, CPU, Memory.**

I. PENDAHULUAN

Perkembangan teknologi dari waktu ke waktu telah mengalami banyak sekali kemajuan, salah satunya pada pengembangan web. Pengembangan web seiring waktu terus mengalami peningkatan dari berbagai aspek, mulai dari kebutuhan tampilan di sisi client, kebutuhan pengembangan di lingkungan server baik dari sisi software maupun hardware, dan kebutuhan manajemen penyimpanan data. Hal ini terus ditingkatkan demi membangun lingkungan pengembangan web yang dinamis dan efektif.

Pencapaian teknologi seperti pada meningkatnya browser modern dan standarisasi HTML 5 menjadi alasan meningkatnya pembuatan aplikasi web yang lebih kompleks. Karena hal itu demand dari pengguna yang mengakses internet mulai mengalami peningkatan dari waktu ke waktu, hal itu dikarenakan semakin majunya teknologi dari suatu aplikasi web yang dikembangkan kebutuhan manusia menjadi semakin terbantu [15]. Oleh karenanya kemampuan *concurrent request* pada web server sangat penting dalam menangani permintaan pada lebih banyak user yang terhubung dan mendapatkan konten yang dibutuhkan dari internet.

Adapun teknologi web server yang masih banyak digunakan saat ini adalah Apache HTTP Server. Statistik tren pada web server per 1 Januari 2023 tercatat sebanyak 32.8% situs web menggunakan Apache HTTP Server [13]. Popularitas Apache HTTP Server terbilang masih tinggi dikarenakan Apache HTTP Server sudah bertahun-tahun menjadi andalan *programmer* dalam menguji dan menjalankan program PHP bersamaan dengan MySQL melalui program *bundling* XAMPP [11]. Karena hal itu orang-orang sudah merasa nyaman dengan Apache HTTP Server dan memilih untuk menggunakan Apache HTTP Server sebagai solusi dalam hosting aplikasi web secara public.

Namun tren web server tidak hanya berhenti pada Apache HTTP Server saja, Node.js hadir sebagai teknologi web server berbasis JavaScript. Namun, Node.js saat ini masih tertinggal jauh dengan Apache HTTP Server dalam hal popularitas. Tren statistik pada penggunaan web server per 1 Januari 2023 tercatat sebanyak 2.0% situs web menggunakan Node.js [13].

Statistik tren pada Apache HTTP Server memang lebih tinggi dibandingkan dengan Node.js. Tetapi hal yang perlu disoroti dari kedua web server adalah kinerja khususnya pada kemampuan *concurrent request* nya. Bagaimana kinerja kedua web server dalam menangani lebih banyak koneksi dari client dengan memroses sejumlah permintaan dan mengirimkan hasilnya kepada client adalah hal yang krusial dibandingkan tingkat popularitas suatu web server.

Dalam konteks web server, *concurrent connection* berperan dalam menerima sejumlah koneksi dari client yang terhubung dengan web server [12]. Sementara client terhubung dengan web server, client akan mengirimkan permintaan (*request*) kepada web server. Sejumlah permintaan yang dikirimkan dari sejumlah client yang sudah terhubung dengan web server adalah pengertian dari *concurrent request* [14]. Client akan tetap terhubung dengan web server sampai permintaan dari client sudah diproses dan diterima dengan sempurna dari Web server.

Semakin besar limit dari jumlah concurrent connection maka semakin besar kemampuan web server dalam menerima setiap koneksi dari client untuk diproses secara bersamaan. Agar mendapatkan kinerja yang optimal jumlah limit *concurrent connection* pada web server ditentukan dari

beberapa faktor diantaranya jenis web server yang digunakan, konfigurasi hardware, serta sumber daya CPU dan memory [12].Namun, bukan berarti dengan memaksimalkan jumlah concurrent connection akan membuat web server semakin cepat dalam menangani permintaan dari client. Arsitektur internal pada web server sangat mempengaruhi efisiensi pemrosesean permintaan atau *concurrent request* dari client.

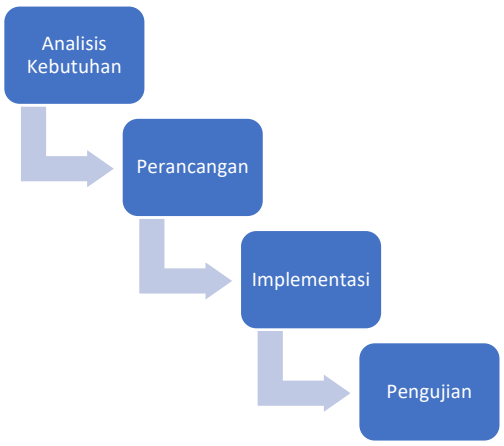
Apache HTTP Server menggunakan pendekatan *process based* pada saat menangani sejumlah *request*. *Process based* merupakan sebuah sistem arsitektur yang secara otomatis melakukan *spawn process (child process)* pada setiap *request* yang masuk pada *parent process* [6]. Apache dapat berjalan dengan baik pada saat menerima *request* tidak begitu banyak, namun pada saat jumlah *request* semakin bertambah Apache akan otomatis membuat process baru lagi untuk menangani *request* tersebut. Sayangnya dengan terus membuat *process* maka akan berdampak pada penggunaan *resource* yang lebih signifikan [1].

Berbeda dengan Apache HTTP Server, arsitektur Node.js menggunakan paradigma *Event-Loop based*. *Event-Loop* merupakan desain arsitektur dalam manajemen proses dimana ketika terdapat *request* yang masuk, node.js otomatis mengantrikannya pada *event queue* untuk dilakukan pemrosesan ke *thread pool* (untuk operasi *blocking*) atau dilakukan secara *single threaded* (untuk operasi *non blocking*) [5]. Setelah diproses, *response* akan dikirimkan kembali ke *client*. Node.js dapat menerima semua *request* untuk ditampung di *event queue* pada *event loop* tanpa perlu menunggu suatu *request* selesai diproses [9]. Hal ini membuat eksekusi menjadi lebih efisien dan tidak tumpang tindih dengan *request* yang semakin bertambah [4].

Penelitian ini bertujuan untuk membuktikan apakah Apache mampu menangani *concurrent request* lebih baik dibandingkan dengan Node.js mengingat statisitk tren Apache lebih tinggi dari dibandingkan Node.js, serta untuk membuktikan bagaimana kinerja arsitektur pada kedua web server dalam menangani *concurrent request*. Pengujian akan dilakukan secara virtualisasi [10]. Tipe *Performance Testing* yang dilakukan pada penelitian ini adalah kinerja skalabilitas [7]-[8]

II. METODE PENELITIAN

Metode pengembangan sistem yang digunakan untuk keberlangsungan penelitian ini menggunakan metode *waterfall*. Metode ini terdiri dari beberapa tahapan sebagai berikut



Gbr 1. Metode Waterfall

A. Analisis Kebutuhan

Analisis kebutuhan merupakan tahapan untuk menganalisis kebutuhan apa saja yang diperlukan untuk selama penelitian berlangsung. Pengujian dilakukan melalui virtual machine (VM) pada platform Google Cloud dengan kebutuhan sebagai berikut :

1). Kebutuhan perangkat keras (Hardware)

Berikut merupakan kebutuhan perangkat keras virtual yang dipergunakan selama penelitian :

- 4 vCPU 2 Core
- 16 GB *memory*

2). Kebutuhan perangkat lunak (Software)

Berikut merupakan kebutuhan perangkat lunak yang dipergunakan selama penelitian :

- Sistem Operasi Debian 11
- Node.js versi 19.8
- Apache HTTP Server versi 2.4
- PHP versi 8.2

B. Perancangan

Sistem dirancang berdasarkan virtualisasi. Setiap VM dibuat untuk menjalankan tugas tersendiri secara terisolasi agar mendapatkan kinerja yang optimal [3]. VM yang terdefinisikan antara lain sebagai berikut :

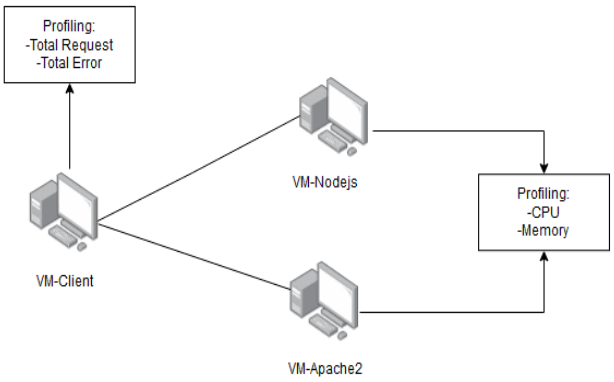
- VM Client yang berfungsi dalam mengirimkan permintaan dan mendapatkan respon dari server.
- VM Server yang berfungsi menerima *request* dan memproses permintaan dari client. VM Server dibuat sebanyak dua *instance,* satu untuk menjalankan Node.js dan satu lagi untuk menjalankan Apache HTTP Server dengan PHP

Penelitian dilakukan dengan menerapkan beberapa model pengujian diantaranya :

1) *Static I/O Test*. Pada tiap *server* dirancang untuk mengirimkan output *hello world* pada *client.*

2) *Computation Test*. Pada tiap *server* dirancang untuk menjalankan algoritma *quick sort* sebanyak 100 angka acak dan diteruskan menuju *client.*
3) *Serving File Test.* Pada tiap *server* dirancang untuk mengirimkan file gambar berukuran 534 KB kepada *client.*

Pengujian dilakukan pada VM Client dengan menggunakan *tool http benchmarking* yaitu wrk. wrk dapat membuat sejumlah client untuk terkoneksi dan mengirimkan permintaan ke web server dengan jumlah client bisa diatur sesuai keinginan pengguna. wrk dapat menampilkan output berupa *total request* yang berhasil dibuat beserta *total error* dari setiap *request* pada setiap koneksi client ke server dalam interval waktu tertentu. Pengumpulan data berupa penggunaan CPU dan *memory* didapatkan melalui VM Server. Sedangakan *total request* dan *total error* didapatkan melalui VM Client melalui *tool* wrk. Pencatatan kinerja CPU dan *memory* dilakukan dengan *software performance monitor* yaitu htop.



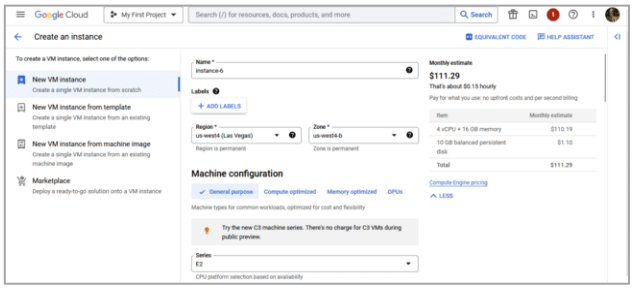
Gbr 2. Skenario Perancangan

C. Implementasi

Tahapan implementasi terbagi menjadi dua tahap diantaranya penyusunan rangkaian virtual machine dan pembuatan program untuk pengujian.

VM Instance dapat dibuat pada menu *Compute Engine* pada *Google Cloud*. Sebelum *VM Instance* dibuat, pengguna perlu mendefinisikan beberapa hal seperti pemilihan *region*, dan pemilihan spesifikasi hardware. Pada penelitian ini VM Client diatur di region asia-southeast1-b sedangkan VM Server untuk Apache HTTP Server dan Node.js diatur di region us-west4-b. Google Cloud menyediakan *machine series* dengan beberapa konfigurasi hardware bawaan sehingga pengguna tidak perlu mendefinisikan kebutuhan hardware secara manual, pada penelitian ini series yang digunakan adalah series e2-standard-4 dengan konfigurasi hardware seperti pemaparan analisis kebutuhan diatas.

Tidak ada konfigurasi *network* tambahan pada penelitian ini. Secara otomatis setiap *VM Instance* yang dibuat akan ter-*assign* dengan IP private dan IP public, IP private nantinya akan dipergunakan untuk pengujian dalam menyambungkan setiap *VM Instance* sesuai perancangan diatas.



Gbr 3. Pembuatan VM Instance pada Google Cloud

Untuk pembuatan program masing masing menggunakan bahasa pemrograman JavaScript untuk Node.js dan PHP untuk Apache HTTP Server. Program dibuat dengan sesederhana mungkin dan semirip mungkin pada kedua bahasa pemrograman untuk menjamin kualitas pengujian kinerja.

Secara default Node.js mampu menerima 1000 *concurrent connection* tanpa dilakukan konfigurasi tambahan. Sedangkan pada Apache HTTP server secara default hanya dapat menerima sebanyak 150 *concurrent connection*. Pada penelitian ini Apache perlu dilakukan konfigurasi agar dapat menyamai variabel dari Node.js yaitu dapat menerima 1000 *concurrent connection*. Dibawah ini adalah konfigurasi default pada Apache yang tersimpan pada module konfigurasi mpm_prefork:

```
StartServers              5
MinSpareServers           5
MaxSpareServers           10
MaxRequestWorkers         150
MaxConnectionsPerChild    0
```

Gbr 3. Konfigurasi default Apache HTTP Server

Dibawah ini adalah konfigurasi 1000 *concurrent connection* untuk Apache HTTP Server

```
StartServers              5
MinSpareServers           5
MaxSpareServers           10
ServerLimit               1000
MaxRequestWorkers         1000
MaxConnectionsPerChild    0
```

Gbr 4. Konfigurasi 1000 concurrent request Apache HTTP Server

Yang perlu disoroti pada konfigurasi diatas adalah *MaxRequestWorkers* dan *ServerLimit*. *MaxRequestWorkers* merupakan jumlah maksimal *connection* untuk diproses secara simultan. Sedangkan *ServerLimit* merupakan jumlah maksimal *server process* untuk melayani setiap permintaan dari client yang sudah terhubung. Jumlah *ServerLimit* harus kurang dari atau sama dengan jumlah *MaxRequestWorkers*.

D. Pengujian

Adapun serangkaian pengujian yang dilakukan untuk memastikan fungsionalitas program pengujian diantaranya:

1) Memastikan setiap VM dapat saling berkomunikasi melalui IP private

2) Memastikan *client* dapat mengirimkan banyak koneksi secara konkuren kepada server
3) Memastikan semua program pengujian dapat berfungsi dengan baik dan dapat melayani *request* dari client

E. Skenario Pengujian

Pada penelitian ini parameter yang dipergunakan selama pengujian adalah jumlah koneksi dan durasi pengujian. Jumlah koneksi client untuk pengujian *concurrent request* ke server selama pengujian berlangsung antara lain 100, 500, dan 1000 *concurrent request*, jumlah tersebut akan diujikan secara bertahap pada tiap model pengujian yang sudah dipaparkan diatas. Untuk waktu pengujian akan diatur sebanyak 20 detik. Selama pengujian berlangsung, penggunaan CPU dan *memory* akan dimonitori pada VM Server. Pengujian akan dilakukan sebanyak 5 kali untuk mendapatkan distribusi *output* pada setiap pengujian.

Pengujian akan dilakukan secara *black box*. Melalui pengujian *black box*, penelitian ini akan berfokus pada pembahasan dan analisis input dan output saja tanpa perlu mengetahui proses dibaliknya.

III. HASIL DAN PEMBAHASAN

A. Hasil Implementasi

Berikut ini dipaparkan hasil implementasi yang sudah direncanakan pada bagian metode penelitian yang selanjutnya akan dilakukan pengujuan dan pengumpulan data.

1). Sumber Kode

Berikut adalah sumber kode JavaScript untuk pengujian kinerja web server Node.js dan PHP untuk pengujian kinerja web server Apache HTTP Server

```
const http = require("http");
const fs = require("fs");

//CREATE SERVER CONNECTION
http.createServer((req, res) => {

//STATIC I/O TEST (HELLO WORLD)

    // res.writeHead(200, { 'Content-Type':
'text/html' });
    // res.end('<h1>Hello World Node.js</h1>')

//COMPUTATION TEST (QUICK SORT)

    function quickSort(array) {
       if (array.length < 2) {
         return array;}
       var pivot = array[0];
       var left = [],
         right = [];
       for (var i = 1; i < array.length; i++) {
         if (array[i] < pivot) {
           left.push(array[i]);
         } else {
           right.push(array[i]);
         }
       }
return
quickSort(left).concat([pivot],quickSort(right));
     }

    var array = [
       89, 47, 62, 36, 78, 22, 51, 11, 100, 94, 57,
63, 42, 76, 31, 85, 99, 55, 5, 17, 68, 28, 45, 71,
8, 60, 83, 79, 48, 92, 43, 12, 61, 23, 87, 54, 3,
72, 37, 80, 98, 26, 65, 53, 16, 25, 95, 39, 74, 20,
50, 91, 69, 40, 75, 30, 86, 4, 58, 19, 77, 13, 70,
24, 46, 14, 96, 7, 49, 64, 38, 56, 82, 15, 84, 9,
21, 34, 97, 27, 67, 52, 66, 90, 41, 6, 35, 10, 73,
18, 81, 44, 29, 59, 88, 2, 32, 1, 33, 93];
     var sortedArray = quickSort(array);
     res.writeHead(200, { "Content-Type":
"text/html" });
     res.end(JSON.stringify(sortedArray));

//SERVING STATIC FILE (IMAGE)

    fs.readFile("static_file.png", (err, data) => {
      res.writeHead(200, { "Content-Type":
"image/png" });
      res.end(data);
    });
})
  .listen(8080, () => {
    console.log("Now listen 127.0.0.1 in Port
8080");
 });
```

Gbr 6. Sumber kode untuk pengujian Node.js

```
<?php
    //STATIC I/O TEST (HELLO WORLD)
    echo '<h1>Hello World Apache PHP';

    //COMPUTATION TEST (QUICK SORT)

    function quickSort($array) {
        if (count($array) < 2) {
          return $array;}
        // Select a pivot element and partition the
array around it
        $pivot = $array[0];
        $left = $right = array();
        for ($i = 1; $i < count($array); $i++) {
          if ($array[$i] < $pivot) {
            $left[] = $array[$i];
          } else {
            $right[] = $array[$i];
          }
        }
        return array_merge(quickSort($left),
array($pivot), quickSort($right));
      }
      $array = array(89, 47, 62, 36, 78, 22, 51, 11,
100, 94, 57, 63, 42, 76, 31, 85, 99, 55, 5, 17, 68,
28, 45, 71, 8, 60, 83, 79, 48, 92, 43, 12, 61, 23,
87, 54, 3, 72, 37, 80, 98, 26, 65, 53, 16, 25, 95,
39, 74, 20, 50, 91, 69, 40, 75, 30, 86, 4, 58, 19,
77, 13, 70, 24, 46, 14, 96, 7, 49, 64, 38, 56, 82,
15, 84, 9, 21, 34, 97, 27, 67, 52, 66, 90, 41, 6,
35, 10, 73, 18, 81, 44, 29, 59, 88, 2, 32, 1, 33,
93);
      $sortedArray = quickSort($array);
      echo json_encode($sortedArray);

      //SERVING FILE TEST (IMAGE)

      header('Content-Type: image/png');
      readfile('static_file.png');
```

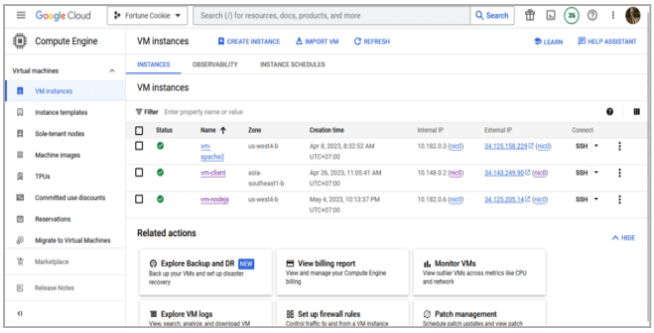
Gbr 7. Sumber kode untuk pengujian Apache

Sumber kode diatas berisikan 3 model pengujian yang dijadikan satu kedalam 1 file index.php dan index.js untuk Apache dan Node.js. Hal ini dikarenakan wrk tidak dapat mengakses route pada web server dan hanya bisa mengakses halaman utama berupa file index saja.

Ketiga model pengujian pada sumber kode diatas tidak akan dijalankan sekaligus. Model pengujian akan dijalankan secara spesifik pada satu model pengujian sehingga terdapat 2 kode model pengujian yang perlu dimatikan (dikomentari).

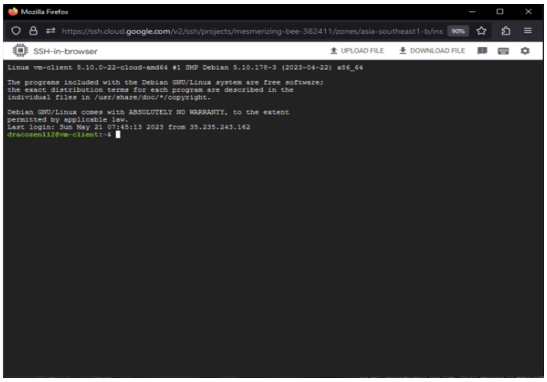
Kode JavaScript diatas perlu memanggil modul inti dari Node.js agar dapat menjalankan web server. Semua kode model pengujian diletakkan di *callback* http.createServer yang berarti kode-kode tersebut akan dijalankan pada setiap *request* yang masuk. Berbeda dengan PHP dimana PHP sejatinya membutuhkan web server untuk menjalankan setiap kode, dan hal ini sudah menjadi tugas dari Apache HTTP Server dalam menerima output dari PHP pada setiap *request* dan mengirimkannya kepada *client*.

2). VM Instance Google Cloud Compute Engine



Gbr 8. List VM Instances untuk pengujian

3 VM Instances diatas dibuat berdasarkan perancangan pada bagian metode penelitian. Seperti yang sudah dijelaskan sebelumnya, IP private akan dipergunakan untuk pengujian meskipun Google Cloud sendiri sudah menyediakan IP external (IP public) secara otomatis setiap VM dinayalakan. VM yang sudah dibuat dapat dioperasikan melalui terminal yang terhubung dengan SSH. Google Cloud sendiri menyediakan fasilitas SSH in Browser dimana pengguna dapat menyambungkan SSH dan mengoperasikan terminal linux langsung di browser.



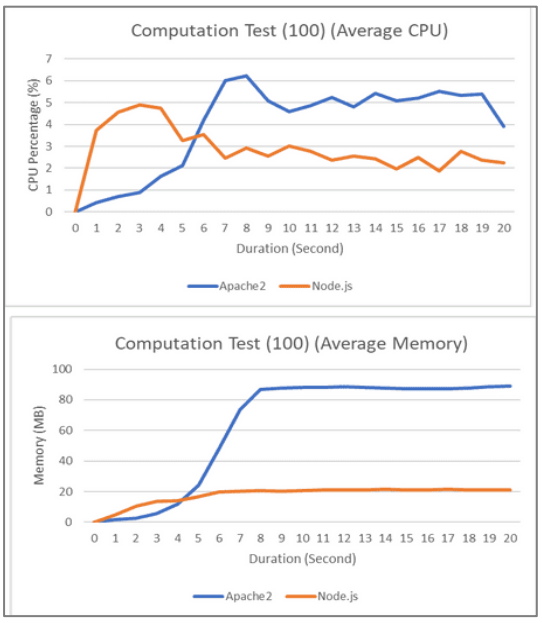
Gbr 9. SSH in Browser

B. Hasil Pengujian

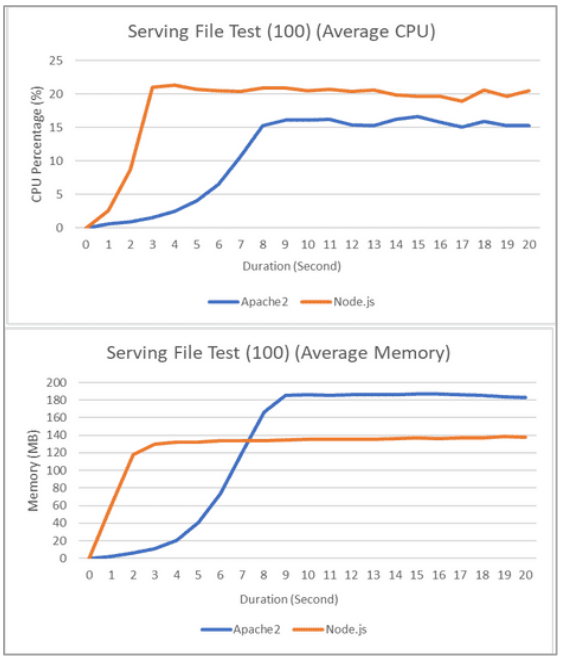
Pemaparan hasil penelitian berikut merupakan kombinasi dari 3 model pengujian dan 3 skenario koneksi yang sudah dipaparkan pada metode penelitian. Total terdapat 9 kali tahap pengujian selama penelitian berlangsung. Diagram CPU *memory*, total timeout, dan error timout yang dipaparkan dibawah ini merupakan hasil dari rata-rata dari 5 iterasi pengujian yang sudah dijalankan. Data CPU yang dipaparkan menggunakan satuan persen (%) sedangkan Data *memory* yang dipaparkan menggunakan satuan (MB).

1). 100 Concurrent Request

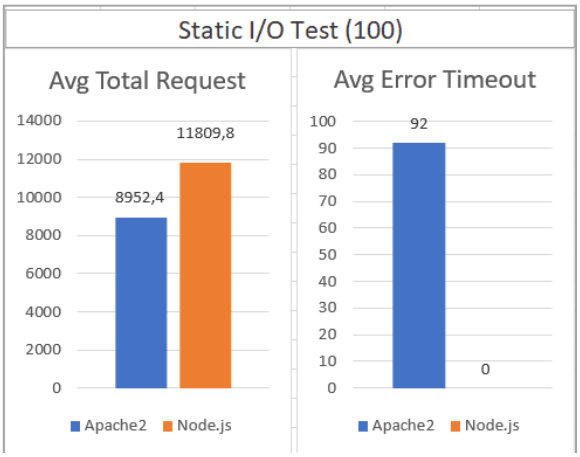
Gbr 9. CPU dan Memory Percobaan Static I/O Test (100)



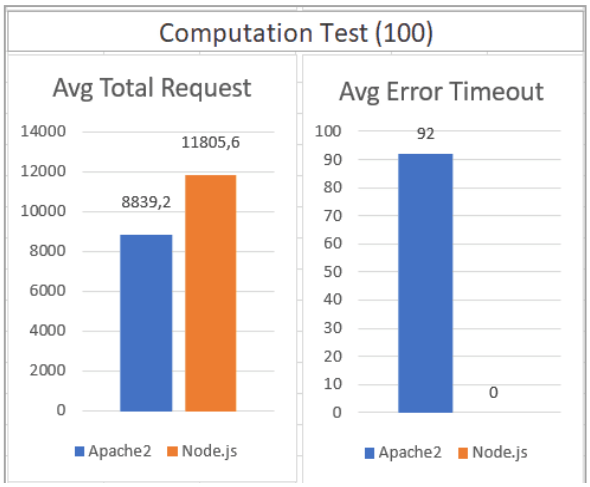
Gbr 10. CPU dan Memory Percobaan Computation Test (100)



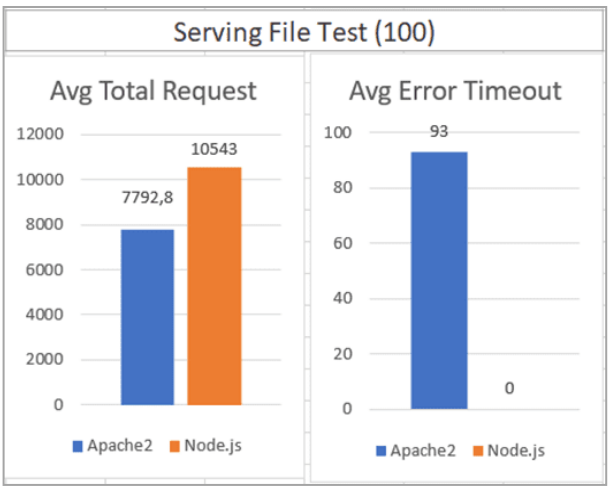
Gbr 11. CPU dan Memory Percobaan Serving File Test (100)



Gbr 12. Total request dan Error Timeout Percobaan Static I/O Test (100)



Gbr 13. Total request dan Error Timeout Percobaan Computation Test (100)



Gbr 14. Total request dan Error Timeout Percobaan Serving File Test (100)

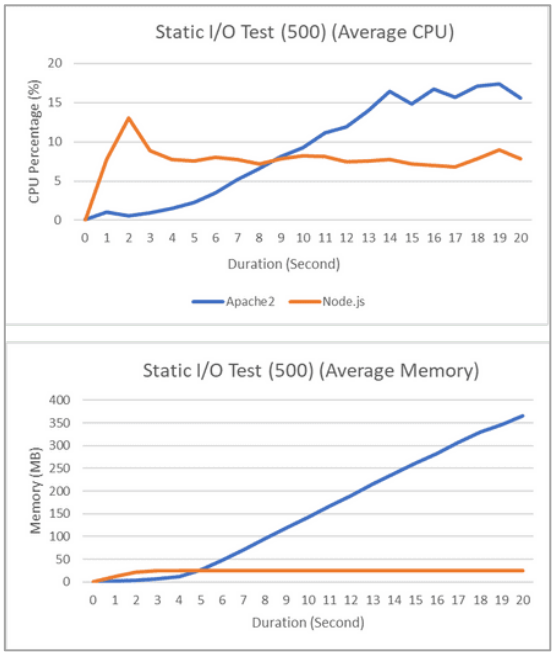
TABEL I
DATA PENGUJIAN 100 CONCURRENT REQUEST

Second	100 Concurrent Request Test											
	Static I/O				Computation				Serving File			
	Apache2		Node.js		Apache2		Node.js		Apache2		Node.js	
	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0,64	1	2,32	2,2	0,42	1,6	3,74	4,8	0,64	2,6	10,7	279,4
2	0,56	2	3,6	2,4	0,7	2,6	4,56	10,6	0,88	6,2	21,7	349,8
3	0,62	3,6	3,98	4	0,88	5,6	4,9	13,4	1,6	10,8	51,2	351,6
4	1,28	8,6	3,52	6,2	1,62	12	4,76	14,2	2,52	20,8	47,08	375,8
5	2,06	18,8	2,12	6	2,12	24	3,26	16,8	4,08	41	47,34	404,8
6	3,68	39	2,26	5,6	4,18	48	3,54	19,8	6,58	72,6	47,46	397,6
7	4,34	61,2	2,16	5,8	6,02	73,4	2,46	20,2	10,7	120,2	45,9	386,2
8	4,64	76,2	2,12	6	6,22	86,8	2,92	20,6	15,3	166,6	47,8	386,4
9	4,28	80	1,74	6,4	5,08	87,8	2,54	20,4	16,08	185,6	46,48	389
10	3,86	80,6	1,78	6,8	4,6	88	3,02	20,6	16,14	186	46,52	380,6
11	3,64	81,6	1,84	6,8	4,88	88	2,78	21	16,28	185,4	47,3	393,2
12	3,7	80,6	1,7	6,8	5,24	88,4	2,38	21	15,4	186	46,4	392
13	3,32	81,2	1,02	6,8	4,8	88	2,54	21,2	15,3	186,2	47,98	399,8
14	4,28	81	1,3	6,8	5,42	87,8	2,44	21,6	16,24	186,4	46,5	386,6
15	3,34	80,4	1,74	6,8	5,1	87,4	1,96	21,2	16,62	186,8	47,58	380,2
16	3,16	80,4	1,7	6,8	5,22	87,4	2,48	21,2	15,78	186,8	46,96	385,2
17	4,08	80,2	1,42	6,8	5,52	87,2	1,86	21,4	15,06	186	48,02	384,4
18	3,36	80,6	1,66	7	5,34	87,8	2,76	21,2	15,94	185,6	46,02	377,2
19	3,06	80,8	1,64	7	5,4	88,6	2,36	21,2	15,32	183,4	47,14	394,6
20	4,02	81	1,3	7,2	3,92	88,8	2,24	21,2	15,28	183	47,62	378,6

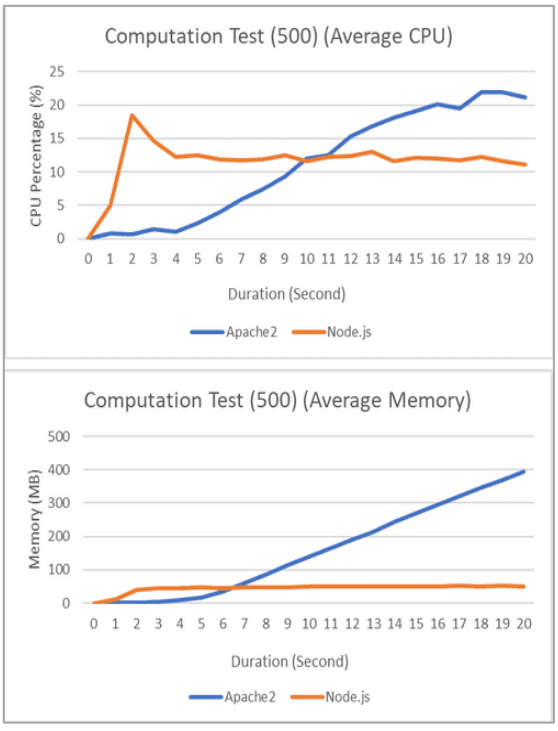
Iteration	100 Concurrent Request Test											
	Static I/O				Computation				Serving File			
	Apache2		Node.js		Apache2		Node.js		Apache2		Node.js	
	Request	Error	Request	Error	Request	Error	Request	Error	Request	Error	Request	Error
1	9133	92	11809	0	8882	92	11875	0	7748	94	10594	0
2	8946	92	11752	0	8726	92	11752	0	7739	94	10590	0
3	9019	92	11803	0	8735	92	11796	0	7741	89	10547	0
4	8832	92	11842	0	8838	92	11802	0	7765	94	10485	0
5	8832	92	11843	0	9015	92	11803	0	7971	94	10499	0

Pada percobaan 100 Concurrent request yang sudah dijalankan, Node.js menunjukkan penggunaan CPU dan memory yang lebih rendah dibandingkan Apache (kecuali pada percobaan Serving File Test (100)). Penggunaan CPU dan memory pada Node.js cenderung mengalami peningkatan signifikan pada detik-detik awal pengujian (sekitar 1 sampai 2 detik) dan mulai mengalami penyesuaian penggunaan CPU dan memory pada detik pengujian setelahnya, dibandingkan dengan Apache yang cenderung mulai mengalami kenaikan secara bertahap pada detik ke-2 sampai 7 pengujian dan stabil di detik setelahnya. Performa pengujian pada Serving File Test (100) menunjukkan beban penggunaan resource yang lebih tinggi dibandingkan dengan Static I/O Test (100) maupun Computation Test (100). Node.js unggul dalam total request dan error timeout dari Apache. Namun pada saat percobaan Serving File Test total request mengalami pengurangan sebanyak ~1000 request dari percobaan Static I/O Test (100) dan Computation Test (100) pada kedua web server. Node.js secara konstan tidak menunjukkan adanya error request pada semua pengujian.

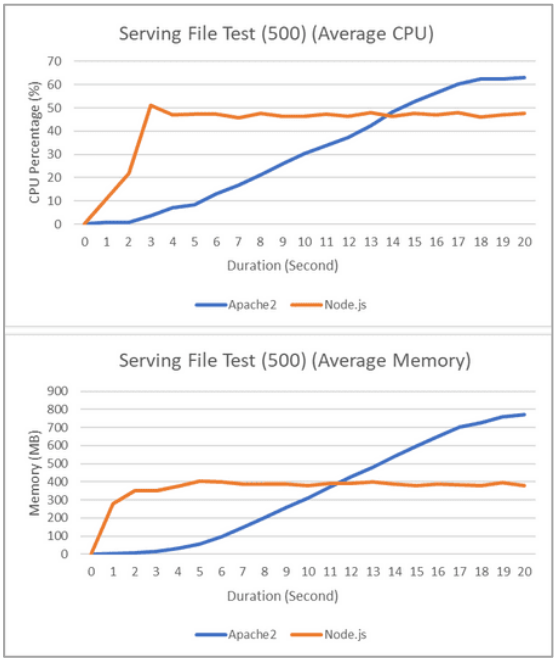
2). 500 Concurrent Request



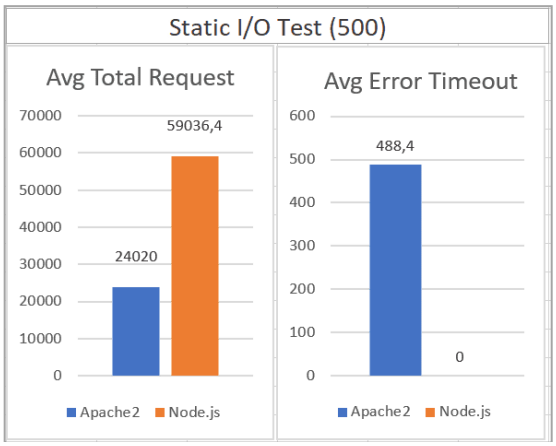
Gbr 15. CPU dan Memory Percobaan Static I/O Test (500)



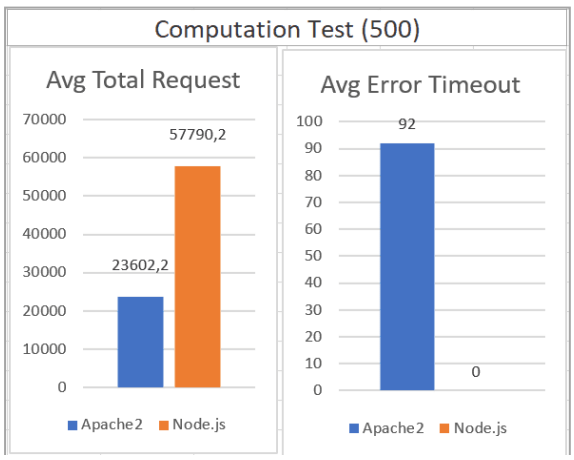
Gbr 16. CPU dan Memory Percobaan Computation Test (500)



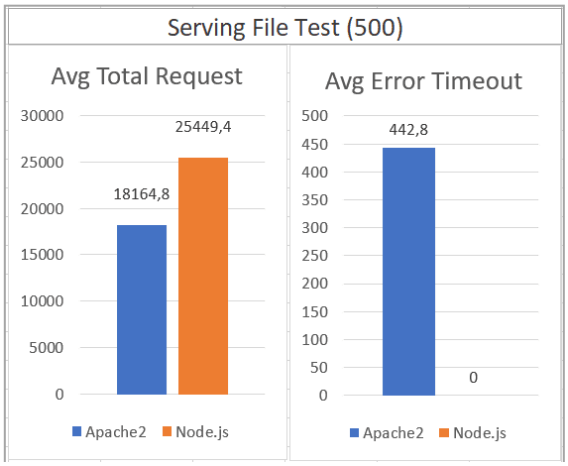
Gbr 17. CPU dan Memory Percobaan Serving File Test (500)



Gbr 18. Total request dan Error Timeout Percobaan Static I/O Test (500)



Gbr 19. Total request dan Error Timeout Percobaan Computation Test (500)



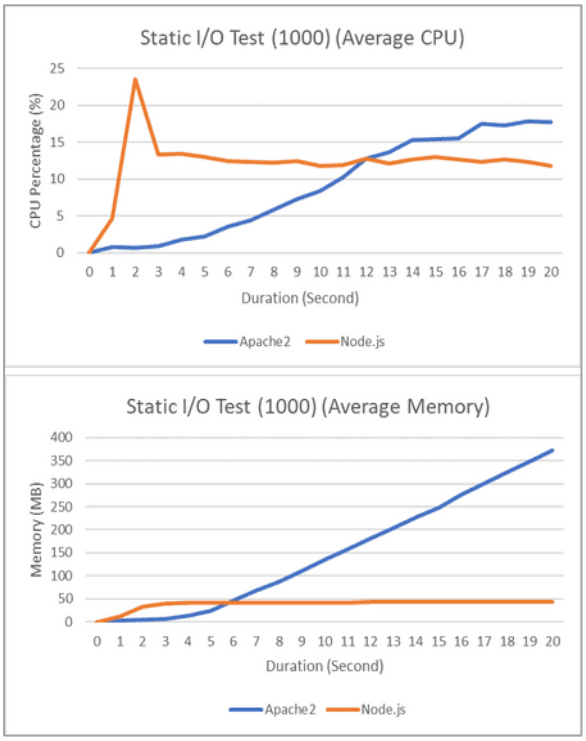
Gbr 20. Total request dan Error Timeout Percobaan Static I/O Test (500)

TABEL II
DATA PENGUJIAN 500 CONCURRENT REQUEST

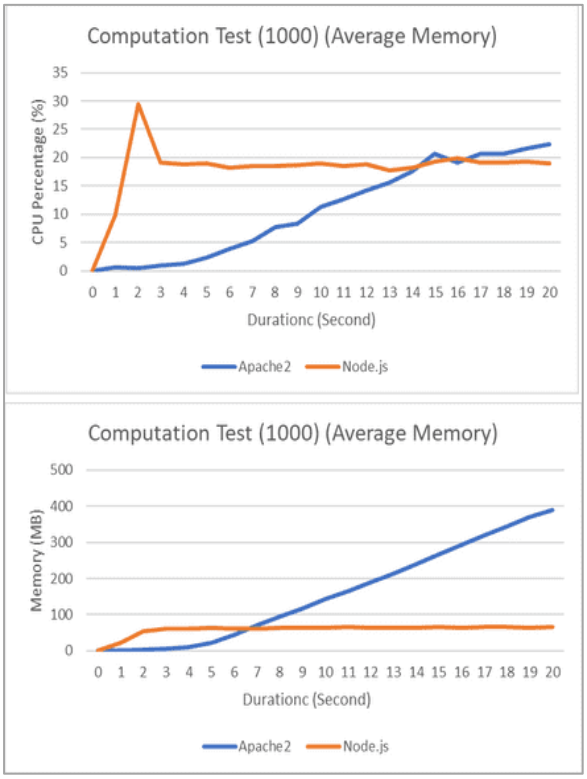
Second	500 Concurrent Request Test											
	Static I/O				Computation				Serving File			
	Apache2		Node.js		Apache2		Node.js		Apache2		Node.js	
	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0,94	2	7,7	11,2	1,04	2,2	4,98	11,4	1,1	3,4	10,7	279,4
2	0,22	3,2	13,06	21,6	0,72	3,2	18,42	39	0,64	6,6	21,7	349,8
3	0,88	5,4	8,88	24,4	1,02	5,4	14,62	44,2	1,18	10,2	51,2	351,6
4	1,16	11,4	7,76	24,6	1,64	11	12,28	44,6	1,74	18,4	47,08	375,8
5	1,84	22,4	7,58	24,2	2,5	22,2	12,46	48,6	2,7	32,6	47,34	404,8
6	3,56	44,8	8	24,4	5,06	44,8	11,88	46	5,1	63,2	47,46	397,6
7	4,78	68	7,76	24,2	6,62	70,2	11,7	48,2	9,9	115,8	45,9	386,2
8	6,1	91,2	7,22	24,6	8,3	95,8	11,92	48	14,24	174,4	47,8	386,4
9	6,86	103,2	7,86	24,6	9,8	112,6	12,46	47,4	18,32	224,4	46,48	389
10	6,3	103,4	8,22	24,4	9,12	115,4	11,64	50,6	23,14	258,4	46,52	380,6
11	5,4	100,4	8,08	24,4	8,32	116	12,24	48,8	23,76	269,6	47,3	393,2
12	6,04	104,6	7,46	24,6	9,58	116,2	12,32	50,4	24,16	269,4	46,4	392
13	4,58	104,6	7,58	24,6	8,82	116,2	12,96	49,6	23,44	269,4	47,98	399,8
14	6,76	104,6	7,78	24,8	8,54	116	11,54	50,8	24,56	270	46,5	386,6
15	5,62	104,6	7,18	24,8	8,8	116	12,12	49,4	23,76	269,8	47,58	380,2
16	5,74	104,6	6,94	24,8	8,8	116,2	12,02	50,6	24,06	269,8	46,96	385,2
17	6,1	104,6	6,82	24,8	8,94	116,4	11,78	51,4	24,36	270,2	48,02	384,4
18	5,14	104,6	7,84	24,8	8,36	116,6	12,28	50,6	23,56	270,6	46,02	377,2
19	5,52	104,6	9	24,8	8,58	116,2	11,6	51,4	24,34	270,4	47,14	394,6
20	6,16	105,4	7,84	24,8	9,74	116,6	11,06	49,8	23,78	268,4	47,62	378,6
Iteration	500 Concurrent Request Test											
	Static I/O				Computation				Serving File			
	Apache2		Node.js		Apache2		Node.js		Apache2		Node.js	
	Request	Error	Request	Error	Request	Error	Request	Error	Request	Error	Request	Error
1	12954	162	58869	0	12591	154	57864	0	11525	149	25686	0
2	12587	154	58914	0	12732	154	57571	0	10851	147	26136	0
3	12577	152	59367	0	12920	162	57231	0	11108	147	26268	0
4	12031	152	58886	0	12365	154	58104	0	10746	146	23080	0
5	12647	154	59146	0	12024	152	58181	0	11105	147	26077	0

Pada percobaan 500 Concurrent request yang sudah dijalankan, Node.js masih unggul di *Total request* pada semua model pengujian dibandingkan dengan Apache dengan tanpa mengalami *error timeout* meskipun *total request* yang didapat Node.js dari pengujian Serving File Test (500) mengalami penurunan *total request* yang signifikan dari 2 percobaan sebelumnya. Tidak bisa dipungkiri bahwa pada pengujian 500 Concurrent request, Apache terus mengalami kenaikan penggunaan CPU sejak detik ke-5 pengujian hingga melampaui titik stabil penggunaan CPU dan *memory* Node.js, Apache mulai menunjukkan titik stabil penggunaan CPU pada detik ke-15 sampai ke-17 pengujian, sedangkan untuk penggunaan *memory* Apache tidak menunjukkan titik stabil sampai detik terakhir pengujian. Apache secara konsisten masih menghasilkan *error timeout* sebesar 430 sampai 490 *error*. Sedangkan Node.js masih bertahan di angka 0 *error*.

3). 1000 Concurrent Request

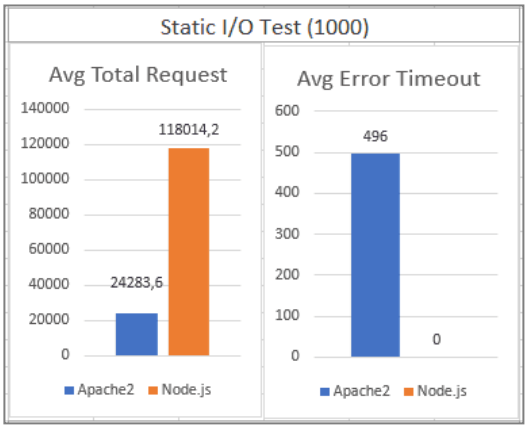


Gbr 21. CPU dan Memory Percobaan Static I/O Test (1000)

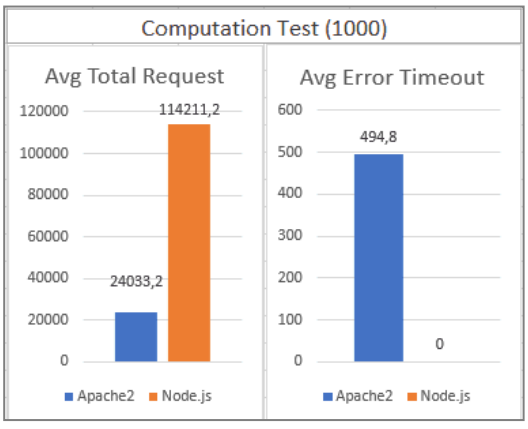


Gbr 22. CPU dan Memory Percobaan Computation Test (1000)

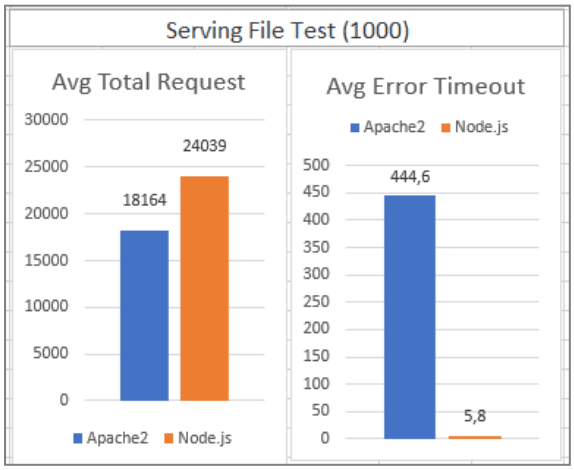
Gbr 23. CPU dan Memory Percobaan Serving File Test (1000)



Gbr 24. Total request dan Error Timeout Percobaan Static I/O Test (1000)



Gbr 25. Total request dan Error Timeout Percobaan Computation Test (1000)



Gbr 26. Total request dan Error Timeout Percobaan Serving File Test (1000)

Tabel III
Data Pengujian 1000 Concurrent Request

Second	1000 Concurrent Request Test											
	Static I/O				Computation				Serving File			
	Apache2		Node.js		Apache2		Node.js		Apache2		Node.js	
	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory	CPU	Memory
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0,72	2,4	4,66	11,4	0,68	1,4	9,88	22,8	0,7	4	16,66	344
2	0,62	4	23,54	32,4	0,56	2,8	29,52	52,8	1	8	42,8	672,4
3	0,86	7	13,28	40,2	1,02	5,2	19,14	61	1,7	12,6	58,34	629,4
4	1,76	13,2	13,44	41,8	1,28	10,8	18,8	60,4	2,1	21,6	56,52	614
5	2,18	23,8	13,04	42,2	2,32	22,4	19	62,8	3,02	40,2	53,2	608,8
6	3,56	47,6	12,44	42	3,94	45	18,2	61,8	6,42	77,6	50,46	615,2
7	4,42	67,4	12,32	41,8	5,22	70,2	18,5	61,2	11,34	127,8	48,34	600,6
8	5,86	87	12,18	41,8	7,66	95	18,48	62,8	15,7	180,2	49,12	596
9	7,28	110	12,42	41,8	8,38	117,6	18,7	64,4	20,44	238,6	48,68	591
10	8,42	134	11,74	41,8	11,3	142	18,92	63,6	24,76	290,6	48,3	591
11	10,24	158,4	11,88	42,2	12,66	165,6	18,5	65	29,5	340,4	49,84	598
12	12,78	180,8	12,82	42,6	14,18	188,4	18,76	63,8	32,9	390	49,16	599,4
13	13,7	203	12,12	42,6	15,6	214,2	17,72	63,4	36,3	445	48,6	612,8
14	15,26	226,2	12,7	42,6	17,62	240,2	18,22	64,2	42,86	501,8	48,5	607,8
15	15,44	248,4	13	42,8	20,6	265,8	19,24	64,8	48,84	559,6	49,56	604
16	15,56	275,2	12,64	43	19,18	292	19,92	62,8	52,86	623,2	49,78	600,8
17	17,52	300	12,28	43	20,74	318,4	19,06	64,8	57,24	661,8	49,88	609,4
18	17,32	324,8	12,62	43	20,62	343,8	19,14	64,8	61	707	49,2	599,6
19	17,84	348,4	12,28	42,8	21,54	369,4	19,32	62,8	61,76	738,6	50	599
20	17,72	372,2	11,82	42,8	22,32	390,4	18,98	65,4	68,98	774,6	48,34	556,4
Iteration	1000 Concurrent Request Test											
	Static I/O				Computation				Serving File			
	Apache2		Node.js		Apache2		Node.js		Apache2		Node.js	
	Request	Error	Request	Error	Request	Error	Request	Error	Request	Error	Request	Error
1	23201	488	118394	0	23086	490	112220	0	17928	446	23413	1
2	24344	488	117783	0	24774	488	113928	0	18811	446	21836	0
3	25335	528	118031	0	25134	520	114671	0	18452	446	23755	9
4	24692	488	117887	0	23441	488	115166	0	17530	439	26171	19
5	23846	488	117976	0	23731	488	115071	0	18099	446	25020	0

Percobaan 1000 *concurrent request* yang sudah dijalankan menunjukkan Node.js mengalami peningkatan kinerja CPU pada semua percobaan 1000 *concurrent request*, hal yang sama terjadi pada konsumsi *memory* dimana Node.js mengalami kenaikan *memory* yang relatif terhadap model pengujian dan skenario koneksi. Hampir sama dengan percobaan 500 *concurrent request* sebelumnya, Apache cenderung masih terus mengalami kenaikan dan tidak mencapai titik stabil, titik stabil hanya terjadi pada penggunaan CPU di percobaan Static I/O Test dan Computation Test (1000). untuk *total request dan error timeout,* Apache hanya menunjukkan perubahan minor dari semua model percobaan di pengujian 500 *concurrent request*. Node.js menunjukkan *total request* yang terlampau tinggi atau sekitar 4.5-5 kali dari *total request* Apache pada percobaan Static I/O Test (1000) dan Computation Test (1000). Pada percobaan Serving File Test (1000) Node.js dan Apache masih menunjukkan *total request* yang hampir sama dengan Serving File Test (500) dimana hanya terdapat selisih sebesar 6000-7000 *request* pada kedua web server. Node.js mendapatkan error untuk yang pertama kali setelah beberapa kali pengujian hanya saja jumlah *error* pada Node.js terbilang sangat rendah dibandingkan dengan Apache yang bisa mencapai hampir 500 *error*.

IV. KESIMPULAN

Pada bagian akhir ini, dipaparkan kesimpulan dari penelitian dan pengujian yang sudah dilakukan terkait kinerja Apache HTTP Server dan Node.js dalam menangani *concurrent request*. Secara keseluruhan Node.js unggul dalam semua skenario *concurrent request* dan model pengujian. Node.js unggul dalam skalabiltas dalam menangani lebih banyak koneksi dengan konsumsi CPU dan *memory* yang cenderung stabil setelah detik ke-2 atau ke-3 pengujian. Node.js nyaris tidak mengalami *error timeout* pada semua skenario pengujian kecuali pada percobaan Serving File Test (1000) yang menghasilkan rata-rata error sebesar 5.8 error, angka ini terbilang jauh lebih rendah dibandingkan dengan Apache HTTP Server yang bisa mencapai 93 sampai 494 error. Paradigma *event-loop based* yang diterapkan pada Node.js menunjukkan adanya kinerja skalabilitas yang lebih baik pada saat menangani lebih banyak permintaan.

Apache menunjukkan kinerja terbaiknya pada skenario 100 *concurrent request* tetapi masih tidak mampu melebihi *total request* yang dihasilkan dari Node.js meskipun penggunaan sumber daya CPU pada Apache masih cenderung masih lebih tinggi. Apache secara konsisten mengalami *error timeout* di semua skenario pengujian dengan total error sebesar 93 sampai hampir menyentuh angka 500. Apache menunjukkan kinerja yang cenderung stagnan dan tidak mengalami banyak perubahan pada saat dilakukan pengujian 500 dan 1000 *concurrent request* dimana penggunaa memory mengalami kenaikan tanpa menunjukkan titik stabil sampai pengujian berakhir, sedangkan pada penggunaan CPU Apache mengalami titik stabil setelah detik ke-15 sampai detik ke-17 (titik stabil CPU tidak terlihat pada pengujian Computation Test (1000) dan Serving File Test (1000)). Arsitketur *Process-based* pada Apache mengalami kesulitan dalam skalabilitas mengingat Apache akan selalu membuat satu *process* untuk melayani setiap *request* yang masuk.

Dari pengujian yang sudah dilakukan dan data yang sudah dipaparkan dapat disimpulkan bahwa meskipun Apache masih jauh populer dikalangan pengembang web dibandingkan Node.js, skalabilitas Apache terbilang rendah dalam menangani lebih banyak permintaan pada semua model pengujian yang sudah dilakukan.

REFERENSI

[1] Gustafsson, A. (2005). Threads without the Pain: Multithreaded programming need not be so angst-ridden. *Queue*, *3*(9), 34-41.
[2] Zeebaree, S. R., Zebari, R. R., & Jacksi, K. (2020). Performance analysis of IIS10. 0 and Apache2 Cluster-based *Web server*s under SYN DDoS Attack. *TEST Engineering & Management*, *83*(March-April 2020), 5854-5863.
[3] RahimiZadeh, Keyvan & Analoui, Morteza & Kabiri, Peyman. (2014). A Performance Model for Evaluating of a Virtualized Computer System.
[4] Chitra, L. P., & Satapathy, R. (2017, February). Performance comparison and evaluation of Node. js and traditional *web server* (IIS). In 2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET) (pp. 1-4). IEEE.
[5] Trillionz, E. (2022) *Why is JavaScript single-threaded and Non-Blocking*, *Elijah Trillionz*. Elijah Trillionz. Available at: https://elijahtrillionz.com/why-is-javascript-single-threaded-and-non-blocking (Accessed: February 23, 2023).
[6] Bradley, S. (2018) Apache-server software with process-based architecture, Vanseo Design. Available at: https://vanseodesign.com/web-design/apache-web-server/ (Accessed: April 10, 2023).
[7] Gillis, A.S. (2023) What is performance testing and how does it work? – TechTarget definition, Software Quality. TechTarget. Available at: https://www.techtarget.com/searchsoftwarequality/definition/performance-testing (Accessed: April 10, 2023).
[8] Hamilton, T. (2023) Performance testing tutorial – types (example), Guru99. Available at: https://www.guru99.com/performance-testing.html (Accessed: April 10, 2023).
[9] Posa, R. (2022) Node JS architecture - single threaded event loop, DigitalOcean. DigitalOcean. Available at: https://www.digitalocean.com/community/tutorials/node-js-architecture-single-threaded-event-loop (Accessed: April 10, 2023).
[10] Brush, K. and Kirsch, B. (2021) What is virtualization? definition from searchservervirtualization, IT Operations. Available at: https://www.techtarget.com/searchitoperations/definition/virtualization (Accessed: 28 May 2023).
[11] Apache Friends. About the XAMPP Project. (n.d.). https://www.apachefriends.org/about.html
[12] "What Is Concurrent Connection Web Server." What Is Concurrent Connection Web Server - Alibaba Cloud, https://www.alibabacloud.com/tech-news/web-server/4l7g2jmvipv-what-is-concurrent-connections-web-server
[13] "Historical Yearly Trends in the Usage Statistics of Web Servers." W3Techs,w3techs.com/technologies/history_overview/web_server/ms/y. Accessed 26 June 2023.
[14] Concurrent requests (2018) KrakenD. Available at: https://www.krakend.io/docs/endpoints/concurrent-requests/ (Accessed: 05 July 2023).
[15] K. Jacksi and S. M. Abass, "Development History of the World Wide Web."