Telematika: Jurnal Informatika dan Teknologi Informasi
Vol. 21, No. 1, Februari 2024, pp.110-121
ISSN: 1829-667X / E-ISSN: 2460-9021
DOI:10.31515/telematika.v21i1.13259

Performance Analysis of FastAPI Framework on Lost Circulation Handling Management Application in Oil Well Drilling

Analisis Performa Framework FastAPI pada Aplikasi Manajemen Penanganan Lost Circulation di Pemboran Sumur Minyak

Andiko Putro Suryotomo[1], Bagus Muhammad Akbar[2], Rochmat Husaini[3]

[1,2,3] Jurusan Informatika, Universitas Pembangunan Nasional Veteran Yogyakarta, Indonesia

[1*]andiko.ps@upnyk.ac.id, [2]bagusmuhammadakbar@upnyk.ac.id, [3]husaini@upnyk.ac.id
**: Penulis korenspondensi* (*corresponding author*)

Informasi Artikel
Received: Januari 2024
Revised: January 2024
Accepted: February 2024
Published: February 2024

Abstract

Purpose*: This study aims to conduct a load testing using JMeter and then analyze the performance of the FastAPI framework on the backend of the lost circulation handling management application in oil well drilling. The developed API receives input in the form of drilling parameter data (daily drilling report) from drilling engineers to be processed by a machine learning model (prediction and classification) through the FastAPI framework. The developed API returns processing data in JSON format.*

Methodology*: Performance measurement is done by conducting load testing simulations using the help of JMeter software. Load testing scenarios are created by varying the number of users and ramp-up time, as well as the method of loading the machine learning model used (normal or preemptive loading). The parameters measured in the test scenario are average execution time, maximum execution time, error percentage, and request throughput.*

Findings*: Load testing on a FastAPI-developed API demonstrated that for compute-heavy tasks like machine learning inference, increasing the number of processor cores and using preemptive model loading led to significantly better performance improvements than changes in processor clock speed or switching from HDD to SSD. Even when simulating a higher user load than initially expected (up to 250 users/threads), FastAPI maintained good response times and a low error rate, remaining below 20%.*

Originality/value/state of the art: *This study result is an information about the performance of the FastAPI framework in the application of lost circulation handling*

ISSN: 1829-667X / E-ISSN: 2460-9021
DOI:10.31515/telematika.v21i1.13259

management in oil well drilling in the deployment phase, not only up to the model testing phase as in previous studies.

Keywords:lost circulation; load testing; FastAPI; JMeter; machine learning
Kata kunci: *lost circulation; load testing;* FastAPI; JMeter; *machine learning*

Abstrak

Tujuan: Penelitian ini bertujuan untuk melakukan load testing menggunakan JMeter dan kemudian menganalisis performa dari framework FastAPI pada *backend* aplikasi manajemen penanganan lost circulation di pemboran sumur minyak. API yang dikembangkan menerima input berupa data parameter pemboran dari *drilling engineer* untuk kemudian diproses oleh model *machine learning* (prediksi dan klasifikasi) melalui framework FastAPI. API yang dikembangkan mengembalikan data pemrosesan dalam bentuk format JSON.
Metodologi: Pengukuran performa dilakukan dengan melakukan simulasi *load testing* menggunakan bantuan perangkat lunak JMeter. Skenario *load testing* dibuat dengan memberikan variasi pada jumlah *user* dan *ramp-up time*, serta cara pemuatan model *machine learning* yang digunakan (normal atau *preemptive loading*). Parameter yang diukur pada skenario pengujian adalah rata-rata waktu eksekusi, waktu eksekusi maksimum, persentase *error*, dan *request throughput.*
Hasil: Pengujian load testing pada API yang dikembangkan dengan FastAPI menunjukkan bahwa untuk *workload* yang *compute heavy* seperti inferensi *machine learning*, peningkatan jumlah *core* prosesor dan penggunaan *preemptive model loading* menghasilkan peningkatan performa yang jauh lebih baik dibandingkan dengan perubahan kecepatan *clock* prosesor atau penggantian jenis *storage* dari HDD ke SSD. Bahkan ketika mensimulasikan beban pengguna yang lebih tinggi dari yang diperkirakan (hingga 250 pengguna/thread), FastAPI tetap mempertahankan waktu respons yang baik dan tingkat *error* yang rendah, yaitu di bawah 20%.
Keaslian/ *state of the art*: Penelitian ini menghasilkan informasi tentang performa framework FastAPI pada aplikasi manajemen penanganan lost circulation di pemboran sumur minyak pada fase *deployment*, tidak hanya sampai pada fase pengujian model seperti pada penelitian-penelitian sebelumnya.

1. Pendahuluan

Lost circulation merupakan suatu fenomena dalam pemboran sumur minyak di mana terjadi hilangnya sebagian atau keseluruhan dari lumpur/fluida *drilling* yang disebabkan oleh masuknya lumpur/fluida *drilling* tersebut ke dalam celah atau pori-pori formasi tanah atau batuan di sekitar sumur minyak yang dibor [1]. Terjadinya *lost circulation* membuat fluida *drilling* terbuang dan tidak menjalankan fungsinya dengan baik, selain itu dalam skala *lost circulation* keseluruhan (*total loss*) hilangnya fluida drilling juga menimbulkan dampak ekonomi untuk perusahaan pemboran dari sisi hilangnya fluida *drilling*[2], menurunnya atau bahkan terhentinya produktivitas sumur minyak (*non-productive time* atau NPT), dan terjadinya *blowout* (keluarnya gas dan minyak bertekanan tinggi yang beresiko menimbulkan ledakan)[3], [4]. Dalam prakteknya, penanganan lost circulation merupakan tindakan *trial and error* [5]dikarenakan banyaknya kemungkinan komposisi dan kombinasi material dan zat kimia yang bisa digunakan. Tindakan penanganan lost circulation juga umumnya lebih bersifat untuk mengurangi tingkat hilangnya fluida drilling bukan untuk menghentikan lost circulation secara total [1][6].

Solusi penanganan *lost circulation* yang memanfaatkan teknologi informasi, khususnya kecerdasan buatan dan *machine learning* telah banyak diteliti[2]. Hameedi, et.al. mengembangkan sebuah model prediksi *machine learning* untuk memprediksi volume *lost circulation*, ECD, dan ROP di formasi Dammam, sebuah zona di ladang minyak Rumaila. Model prediksi yang dikembangkan memanfaatkan kumpulan data yang lebih besar dan metode statistik sistematis, mengungguli model sebelumnya, menawarkan prediksi yang lebih ketat dan mengoptimalkan faktor operasional seperti ukuran *nozzle* untuk ROP yang optimal [4]. Metode klasik machine learning seperti SVM, random forest, dan K-NN digunakan untuk memprediksi laju *lost circulation* selama pengeboran pada sebuah *fractured formations*. Dengan menganalisis data dari tujuh sumur, model dibuat dan divalidasi pada sumur kedelapan. Model K-NN menghasilkan kinerja terbaik, dengan koefisien korelasi 0,90 dan RMSE 0,17, yang menunjukkan akurasinya yang kuat dalam memprediksi laju *lost circulation* berdasarkan parameter mekanis permukaan dan pengukuran volume sumur [7]. Penelitian lain menggunakan *Mixture Density Network* (MDN) dengan lima sub-Gaussian model untuk memprediksi terjadinya *lost circulation*. Metode yang diusulkan menjadi metode yang terbaik dengan nilai Negative Log-Likelihood sebesar 3,143 [3]. Model *machine learning* dan *deep learning* dibandingkan dalam memprediksi kelas *lost circulation* dari data yang *imbalance* menghasilkan model machine learning berbasis *tree*, khususnya *random forest*, mengungguli model *deep learning* dalam hal akurasi dan kecepatan. Gabungan model *random forest*, Adaboost, dan *decision tree* direkomendasikan untuk prediksi kehilangan fluida yang handal [8]. Kemudian pendekatan berbasis *machine learning* digunakan untuk memprediksi kejadian *lost circulation* selama fase well planning (perencaanaan pemboran sumur). Metode yang disarankani berfokus pada persiapan data, pembelajaran mesin, dan penerapan model, dengan memanfaatkan atribut dari *well offset*, *drilling log*, dan data seismik. Meskipun spesifisitasnya rendah untuk kejadian dengan probabilitas tinggi, model ini unggul dalam memprediksi zona dengan probabilitas rendah. Pendekatan ini dapat mengurangi risiko kehilangan sirkulasi hingga 95%, sehingga memungkinkan perencanaan lintasan sumur dan alokasi sumber daya yang lebih baik selama proses konstruksi sumur[9]. Seluruh penelitian tersebut memiliki fokus penelitian hanya sampai pada tahap pengujian model, belum dilakukan analisi sampai pada *deployment* atau penggunaan model pada pemboran sumur minyak sesungguhnya. Variabel yang diprediksi atau

diklasifikasikan juga terbatas pada variabel terjadi atau tidaknya *lost circulation* atau prediksi material yang cocok untuk mengatasi *lost circulation* yang terjadi.

FastAPI merupakan salah web framework berbasis Python yang digunakan untuk membangun sebuah *Application Programming Interface* (API)[10]. FastAPI menggunakan konsep *Asynchronous Server Gateway Interface* (ASGI) yang memungkinkan *endpoint* API pada FastAPI untuk dipanggil dan mengeksekusi perintah secara asinkron[11]. FastAPI sendiri merupakan sebuah framework API yang sesuai untuk workload machine learning, terutama untuk menghasilkan sebuah platform *Machine Learning as a Service* (MlaaS)[11][12]. Pada penelitian ini akan dilakukan load testing dan analisis performa dari API berbasis FastAPI yang merupakan backend dari sebuah aplikasi manajemen penanganan *lost circulation*. Aplikasi ini mampu memberikan alternatif solusi untuk material yang sesuai untuk menangani *lost circulation* yang terjadi beserta dengan parameter penyertanya (komposisi material, besaran dari masing-masing komposisi material, NPT, dan lain sebagainya).

Load testing akan dilakukan dengan menggunakan JMeter, sebuah aplikasi simulasi load testing *open source* (Apache)[13]. Jmeter mampu mensimulasikan kondisi pengguna secara real time dengan cara mengatur *ramp up time* pengguna virtual mereka secara sekuensial[14] atau secara random sehingga menyerupai kondisi akses di dunia nyata[15]. Pada penelitian ini akan dilakukan pada empat jenis Virtual Private Server (VPS) yang memiliki spesifikasi dan konfigurasi hardware yang berbeda. Skenario pengujian disesuaikan dengan keadaan penggunaan aplikasi di dunia nyata (aplikasi telah digunakan di perusahaan perminyakan milik negara di seluruh Indonesia), dengan proyeksi minimal pengguna konkuren sebanyak 100 pengguna dan maksimal sebanyak 250 pengguna setiap waktunya.

2. Metode dan Perancangan

Pada metode dan perancacngan akan dibahas tentang spesifikasi API, spesifikasi VPS yang digunakan, dan skenario *load testing* pada JMeter.

2.1. Spesifikasi *Application Programming Interface* (API) yang akan diuji

API yang akan diuji merupakan sebuah *backend* dari aplikasi manajemen penanganan *lost circulation* yang dibuat dengan menggunakan FastAPI. API menerima data input dari pengguna berupa parameter *drilling* yang kemudian akan diproses oleh model *machine learning* pada API. Hasil dari pemrosesan tersebut akan dikembalikan dalam bentuk format JSON.

Terdapat 18 parameter drilling sebagai input dari API dan 39 parameter output yang dihasilkan dari 42 model machine learning yang digunakan. Parameter input dari pengguna tersebut disampaikan pada **Tabel 1**, sedangkan parameter output dari model *machine learning* dapat dilihat pada **Tabel 2**.

Tabel 1. Parameter input dari model machine learning

Nama parameter	Satuan
Well diameter	inch
Well depth	Meter
Casing depth	Meter
Depth at LC Occurred	Meter

ISSN: 1829-667X / E-ISSN: 2460-9021
DOI:10.31515/telematika.v21i1.13259

Pump rate	*Gallon per minute* (GPM)
Loss rate	*Barrell per minute* (BPM)
Operation at LC occurred	Tanpa satuan (kategorikal)
Fill on bottom fluid type	Tanpa satuan (kategorikal)
Fill on bottom fluid Density	SG
Fill on bottom fluid YP	SG
Fill on bottom fluid PV	SG
Fill on bottom fluid 10 min Gel Strength	*lbs*/100 ft^2
Mechanical support	Tanpa satuan (kategorikal)
Formation collapse	Tanpa satuan (kategorikal)
Rock type	Tanpa satuan (kategorikal)
Purpose of treatment	Tanpa satuan (kategorikal)
Mud density	SG
Mud temperature	Fahrenheit

Tabel 2. Parameter output dari model *machine learning*

No	Parameter Output
1	Nozzle depth (m)
2	OE depth (m)
3	Volume of cement slurry (bbl)
4	Size of bit nozzle
5	Size of OE
6	Plug Type
7	BHST Test (deg F)
8	BHCT Test (deg F)
9	Density Cement (SG)
10	Density of LCM/BDO/LDA mud (SG)
11	Water Requirement Cement (GPS)
12	PV Cement
13	YP Cement
14	bacaan rpm 300
15	bacaan rpm 200
16	bacaan rpm 100
17	bacaan rpm 60
18	bacaan rpm 30
19	bacaan rpm 6
20	bacaan rpm 3
21	Fluid Loss
22	Free water nil?
23	Initial BC
24	Thickening time (70BC)
25	Fill on bottom fluid type ahead

No	Parameter Output
26	Fill on bottom fluid type behind
27	Fill on bottom fluid Density ahead (SG)
28	Fill on bottom fluid Density behind (SG)
29	Fill on bottom fluid Volume ahead (bbl)
30	Fill on bottom fluid Volume behind (bbl)
31	Fill on bottom fluid type SPACER ahead (SG)
32	Fill on bottom fluid type SPACER behind (SG)
33	Fill on bottom fluid type SPACER ahead (bbl)
34	Fill on bottom fluid type SPACER behind (bbl)
35	Fill on bottom fluid ahead (ppb)
36	Spacer Density ahead (SG)
37	Spacer Density behind (SG)
38	Spacer ahead (bbl)
39	Spacer behind (bbl)

2.2. Spesifikasi *Virtual Private Server* (VPS) yang akan diuji

Spesifikasi Virtual Private Server (VPS) yang digunakan memiliki variasi dari segi jumlah core pemroses, ukuran RAM, dan jenis storage yang digunakan. Spesifikasi VPS tersebut dapat dilihat pada Tabel 3.

Tabel 3. Spesifikasi VPS yang diuji

ID/Nama VPS	Spesifikasi		
	Jumlah *core* & *clock speed* prosesor	Ukuran RAM	Ukuran dan jenis *storage*
Case 1	Dual (2) *core*, 3,6 Ghz	8 GB	40 GB, Harddisk (HDD)
Case 2	Dual (2) *core*, 3,6 Ghz	8 GB	40 GB, SSD
Case 3	Quad (4) *core*, 3,2 Ghz	8 GB	40 GB, Harddisk (HDD)
Case 4	Quad (4) *core*, 3,2 Ghz	8 GB	40 GB,SSD

Variasi spesifikasi VPS ini dimaksudkan untuk mengukur dan menganalisis pengaruh kecepatan storage (SSD versus HDD) yang akan berpengaruh pada *workload* yang *I/O heavy* (seperti ketika API memuat model *machine learning*)[16] dan pengaruh kecepatan dan jumlah *core* pada CPU yang akan berpengaruh pada *workload* yang *compute heavy* (seperti ketika API memproses data input dengan model *machine learning*).

2.3. Skenario load testing dengan JMeter

Parameter load testing pada JMeter yang akan digunakan dalam konfigurasi load testing penelitian ini adalah sebagai berikut.

a. Jumlah *thread*, yaitu simulasi jumlah *request* (atau pengguna) dalam sebuah skenario pengujian. Pada penelitian ini akan digunakan empat macam jumlah *thread* yaitu 100, 250, 500, dan 1000 thread.
b. *Ramp up period* (dalam detik), yaitu waktu yang dibutuhkan oleh JMeter untuk menjalankan semua *thread* yang ingin disimulasikan. Waktu dieksekusinya sebuah thread bisa dibuat secara acak atau secara bertahap berurutan[17]. Dalam penelitian ini akan digunakan *ramp up period* sebesar 10, 30, 60, dan 100 detik.
c. *Loop count* yaitu banyaknya jumlah suatu skenario pengujian dilakukan secara berulang. Akan digunakan *loop count* sebanyak 10.

Kemudian untuk parameter *load testing* yang akan diukur dengan JMeter adalah sebagai berikut.

a. *Average execution time* (dalam milidetik/milisekon) yaitu waktu rata-rata yang dibutuhkan oleh sebuah *thread* dimulai dari mengirim *request* sampai mendapatkan *response* dari API.
b. *Maximum execution time* (dalam milidetik/milisekon) yaitu waktu terlama yang dibutuhkan oleh sebuah *thread* dimulai dari mengirim *request* sampai mendapatkan *response* dari API.
c. *Error percentage* (dalam persen/%) yaitu persentase dari *request* yang mengembalikan response kesalahan (*error*) atau tidak mengembalikan response sama sekali (*Request Time Error*/RTO).
d. *Throughput* (dalam request / detik) yaitu banyaknya *request* yang bisa diproses oleh API dalam satu satuan waktu.

Akan diujikan tiga macam rangkaian pengujian (*load testing*) yaitu sebagai berikut.

a. Rangkaian pengujian pertama akan menganalisis performa API berdasarkan konfigurasi jumlah *core* pada prosesor VPS dan jenis media *storage* dari masing-masing VPS.
b. Rangkaian pengujian kedua akan menganalisis sensitivitas workload API pada perubahan kecepatan prosesor (*clock speed*) pada VPS.
c. Rangkaian pengujian ketiga akan menganalisis penerapan *preemptive model loading* pada API.

3. Hasil dan Pembahasan

3.1. Analisis performa API berdasarkan konfigurasi jumlah core dan jenis media storage VPS

Dari rangkaian pengujian pertama, penggunaan CPU dengan *core* pemroses yang lebih banyak mampu memberikan peningkatan paling tinggi pada parameter *Throughput* (request/sec). Pada perbandingan pengujian Case 1 dan Case 3 terjadi peningkatan *Throughput* sebesar 109,03 % dan pada Case 2 dan Case 4 terjadi peningkatan *throughput* sebesar 64,2 % (ditampilkan pada **Tabel 5**).

ISSN: 1829-667X / E-ISSN: 2460-9021
DOI:10.31515/telematika.v21i1.13259

Rangkaian pengujian pertama juga menunjukkan bahwa penggunaan *storage* dengan jenis SSD memberikan peningkatan signifikan pada parameter throughput. Peningkatan *throughput* terbesar yaitu 57,45% terjadi pada pengujian pada Case 1 dan Case 2 (ditampilkan pada **Tabel 5**). Pada pengujian Case 3 dan Case 4 peningkatan *Throughput* karena perubahan jenis media penyimpanan hanya sebesar 24,33 %. Hal ini terjadi karena pada Case 3 dan Case 4 tersebut, *resource* komputasi yang dimiliki oleh prosesor quad core bisa dimanfaatkan oleh framework FastAPI untuk menjalankan eksekusi perintah secara *concurrent* atau bahkan paralel (ditampilkan pada **Tabel 5**).

Dua fakta di atas menunjukkan bahwa workload inferensi model prediksi dan klasifikasi machine learning yang digunakan merupakan *workload* yang lebih *compute heavy* atau dalam kata lain workload yang lebih banyak memanfaatkan *resource* CPU daripada *storage* dan RAM.Hasil rangkaian pengujian pertama dapat diihat pada **Tabel 4.**

Tabel 4. Hasil load testing untuk setiap case/VPS

Case 1					
Jumlah thread	**Ramp up period (s)**	**Avg. Execution time (ms)**	**Max. execution time (ms)**	**Error %**	**Throughput (request / s)**
100	10	5210	7034	14	2.7
	30	4802	6483	13	2.7
	60	3524	4757	12	2.6
	100	2979	4022	13	2.6
250	10	6773	9144	15	2.8
	30	6243	8428	14	2.9
	60	4581	6184	15	2.7
	100	3873	5229	13	2.7
500	10	8597	11606	16	2.7
	30	7923	10696	16	2.6
	60	5815	7850	14	2.7
	100	4915	6635	14	2.7
1000	10	9118	12309	17	2.2
	30	8404	11345	15	2.1
	60	6167	8325	15	2.1
	100	5213	7038	15	2.1

Case 2					
Jumlah thread	**Ramp up period (s)**	**Avg. Execution time (ms)**	**Max. execution time (ms)**	**Error %**	**Throughput (request / s)**
100	10	4848	6545	13	4.1
	30	4681	6319	11	4.1
	60	3113	4203	10	4.1
	100	2815	3800	13	4
250	10	6302	8508	13	4.3
	30	6085	8215	13	4.3
	60	4047	5463	13	4.2
	100	3660	4941	11	4.1
500	10	7999	10799	15	4.1
	30	7724	10427	15	4
	60	5136	6934	14	4

ISSN: 1829-667X / E-ISSN: 2460-9021
DOI:10.31515/telematika.v21i1.13259

Case 2					
Jumlah thread	**Ramp up period (s)**	**Avg. Execution time (ms)**	**Max. execution time (ms)**	**Error %**	**Throughput (request / s)**
	100	4645	6271	13	3.9
1000	10	8484	11453	17	3.8
	30	8192	11059	15	3.7
	60	5448	7355	13	3.7
	100	4926	6650	14	3.6

Case 3					
Jumlah thread	**Ramp up period (s)**	**Avg. Execution time (ms)**	**Max. execution time (ms)**	**Error %**	**Throughput (request / s)**
100	10	4845	6202	10	5.6
	30	4466	5716	8	5.5
	60	3277	4195	9	5.5
	100	2770	3546	12	5.3
250	10	6299	8063	12	5.4
	30	5806	7432	11	5.4
	60	4260	5453	10	5.3
	100	3602	4611	9	5.2
500	10	7995	10234	12	5.3
	30	7368	9431	14	5.2
	60	5408	6922	12	5.1
	100	4571	5851	11	5
1000	10	8480	10854	15	5
	30	7816	10004	12	5.1
	60	5735	7341	10	5.4
	100	4848	6205	12	5.3

Case 4					
Jumlah thread	**Ramp up period (s)**	**Avg. Execution time (ms)**	**Max. execution time (ms)**	**Error %**	**Throughput (request / s)**
100	10	4118	4900	9	7.2
	30	3796	4517	8	7.1
	60	2785	3314	8	7
	100	2355	2802	12	7
250	10	5354	6371	12	7
	30	4935	5873	10	6.8
	60	3621	4309	10	7
	100	3062	3644	8	6.9
500	10	6796	8087	11	6.5
	30	6263	7453	13	6.6
	60	4597	5470	12	6.6
	100	3885	4623	11	6.4
1000	10	7208	8578	14	6.1
	30	6644	7906	12	5.7
	60	4875	5801	10	5.6
	100	4121	4904	11	5.7

Tabel 5 perbandingan perubahan parameter antar Case/VPS pada pengujian pertama.

Case 1 vs Case 2 (perbandingan jenis media *storage*)			
Improvement Avg. Execution time (%)	**Improvement Max. execution time (%)**	**Penurunan Error percentage (%)**	**Improvement Throughput (%)**
6.66	6.66	8	57.45
Case 3 vs Case 4 (perbandingan jenis media *storage*)			
Improvement Avg. Execution time (%)	**Improvement Max. execution time (%)**	**Penurunan Error percentage (%)**	**Improvement Throughput (%)**
15	20.98	4.49	24.33
Case 1 vs Case 3 (perbandingan jumlah *core* pada prosesor)			
Improvement Avg. Execution time (%)	**Improvement Max. execution time (%)**	**Penurunan Error percentage (%)**	**Improvement Throughput (%)**
7	11.82	22.72	109.03
Case 2 vs Case 4 (perbandingan jumlah *core* pada prosesor)			
Improvement Avg. Execution time (%)	**Improvement Max. execution time (%)**	**Penurunan Error percentage (%)**	**Improvement Throughput (%)**
15.20344	25.25573	19.91725	64.20662

3.2. Analisis sensitivitas workload API pada perubahan kecepatan prosesor (*clock speed*) VPS

Rangkaian pengujian selanjutnya bertujuan untuk mengukur sensitivitas *workload compute heavy* dengan perubahan kecepatan prosesor pada VPS. Case 4 diambil sebagai contoh karena dari seluruh Case, Case 4 memiliki spesifikasi VPS yang paling tinggi.

Dengan peningkatan base core clock sebesar 400 Mhz (menjadi 3,6 Ghz) didapatkan rata-rata peningkatan throughput sebesar 20%, penurunan rata-rata error percentage sebesar 5,5 %, penurunan maximum execution time sebesar 6 %, dan penurunan average execution time sebesar 6 %.

Tabel 6. perbandingan perubahan parameter antar Case/VPS pada pengujian kedua

Case Case 4 (Perubahan *core clock* sebesar 400 Mhz pada prosesor)			
Improvement Avg. Execution time (%)	**Improvement Max. execution time (%)**	**Penurunan Error percentage (%)**	**Improvement Throughput (%)**
6	6	5.5	20

3.3. Analisis *preemptive model loading* pada API

Preemptive model loading adalah perintah kepada API untuk memuat seluruh model yang sudah dilatih tanpa memperdulikan apakah model tersebut akan langsung digunakan atau tidak. Untuk kondisi di mana VPS atau server terbatas pada resource I/O (RAM, *network*, *storage*) hal ini sangat disarankan supaya tidak terjadi *bottleneck* setiap kali model dipanggil oleh API. *Tradeoff* yang mungkin muncul dari metode ini adalah meningkatnya penggunakan *memory*/RAM dan waktu inisialisasi API yang lebih lama.

Preemptive model loading memberikan peningkatan parameter *Throughput* yang jauh lebih tinggi jika dibandingkan dengan hasil pada rangkaian pengujian pertama. Peningkatan *throughput* sebesar 90,42 % didapatkan dari perbandingan Case 3 pada pengujian pertama dan kedua, serta angka peningkatan sebesar 110,18 % didapatkan dari perbandingan Case 4 pada pengujian pertama dan kedua. Selain itu angka parameter *Error Percentage* juga mengalami

perbaikan, penurunan angka *Error Percentage* sebesar 40,74 % didapatkan dari perbandingan Case 3 pada pengujian pertama dan kedua, serta penurunan sebesar 44,15 % didapatkan dari perbandingan Case 4 pada pengujian pertama dan kedua.

Peningkatan performa *Throughput* dan penurunan *Error Percentage* yang didapatkan dari implementasi *preemptive model loading* ini disertai dengan peningkatan penggunaan *memory*/RAM pada VPS sebesar 21% pada Case 3 (dari 251 MB menjadi 303 MB) dan 19,8% pada case 4 (dari 260 MB menjadi 311 MB).

4. Kesimpulan dan Saran

Pengujian *load testing* pada API yang dikembangkan dengan menggunakan *framework* FastAPI menunjukkan bahwa pada workload yang lebih *compute heavy* (seperti inferensi dengan model *machine learning*) kombinasi faktor jumlah core pada prosesor dan preemptive model loading menghasilkan peningkatan performa yang jauh lebih besar dibandingkan dengan perubahan kecepatan clock prosesor dan perubahan jenis media storage (dari HDD ke SSD). Selain itu, dengan simulasi jumlah pengguna yang melebihi spesifikasi awal (maksimum 250 user/thread), FastAPI tetap mampu memberikan waktu respons yang baik dengan tingkat error yang rendah (di bawah 20%).

Belum adanya perbandingan dan analisis performa FastAPI dengan framework API berbasis Python yang lain bisa menjadi fokus pada penelitian selanjutnya. Selain itu pengujian keamanan API (pada deployment, API yang diujikan pada penelitian ini sudah *comply* dengan standar keamanan OWASP) dengan menggunakan standar-standar keamanan jaringan, dan pengaruh tingkat dan kompleksitas keamanan jaringan dengan performa FastAPI menarik untuk diteliti lebih lanjut.

Daftar Pustaka

[1] A. Lavrov, “Lost circulation: Mechanisms and solutions,” *Lost Circ. Mech. Solut.*, pp. 1–252, Mar. 2016, doi: 10.1016/C2015-0-00926-1.

[2] H. Elmousalami and I. Sakr, “Artificial intelligence for drilling lost circulation: A systematic literature review,” *Geoenergy Sci. Eng.*, vol. 239, p. 212837, Aug. 2024, doi: 10.1016/J.GEOEN.2024.212837.

[3] H. Pang, H. Meng, H. Wang, Y. Fan, Z. Nie, and Y. Jin, “Lost circulation prediction based on machine learning,” *J. Pet. Sci. Eng.*, vol. 208, p. 109364, Jan. 2022, doi: 10.1016/J.PETROL.2021.109364.

[4] A. T. Al-Hameedi *et al.*, “Using Machine Learning to Predict Lost Circulation in the Rumaila Field, Iraq,” Oct. 2018, doi: 10.2118/191933-MS.

[5] P. Pauhesti, A. Sembiring, M. Djumantara, L. Samura, C. Rosyidan, and K. Kunci, “EVALUASI PENANGGULANGAN LOST CIRCULATION LAPANGAN X,” *PETRO J. Ilm. Tek. Perminyakan*, vol. 12, no. 2, pp. 89–97, May 2023, doi: 10.25105/PETRO.V12I2.14383.

[6] X. Hou *et al.*, “Lost Circulation Prediction in South China Sea using Machine Learning and Big Data Technology,” *Proc. Annu. Offshore Technol. Conf.*, vol. 2020-May, May

2020, doi: 10.4043/30653-MS.

[7] A. Alsaihati, M. Abughaban, S. Elkatatny, and D. Al Shehri, "Application of Machine Learning Methods in Modeling the Loss of Circulation Rate while Drilling Operation," *ACS Omega*, vol. 7, no. 24, pp. 20696–20709, Jun. 2022, doi: 10.1021/ACSOMEGA.2C00970/ASSET/IMAGES/LARGE/AO2C00970_0009.JPEG.

[8] D. A. Wood, S. Mardanirad, and H. Zakeri, "Effective prediction of lost circulation from multiple drilling variables: a class imbalance problem for machine and deep learning algorithms," *J. Pet. Explor. Prod. Technol.*, vol. 12, no. 1, pp. 83–98, Jan. 2022, doi: 10.1007/s13202-021-01411-y.

[9] V. Guillot, A. Ruzhnikov, M. Corona, and F. Karpfinger, "Machine Learning Prediction of the Lost Circulation Events at the Well Planning Stage," *Offshore Technol. Conf. Asia, OTCA 2024*, Feb. 2024, doi: 10.4043/34764-MS.

[10] C. V. Suresh babu, V. Surendar, E. Sriram, and S. Subhash, "Web-Based Deep Learning Model for Zero Day Vulnerability Detection using FastAPI," *2024 Int. Conf. Adv. Data Eng. Intell. Comput. Syst.*, pp. 1–6, Apr. 2024, doi: 10.1109/ADICS58448.2024.10533540.

[11] M. Raihan, S. Putra Pamungkas, M. Nurul Huda, D. A. Fauzan, A. Hilal Itsna, and M. Al-Hijri, "Sistem Klasifikasi Otomatis Dengan Konsep Machine Learning As A Service (MLaaS) Pada Kasus Pesan Berindikasi Cyberbullying," *Ilk. J. Comput. Sci. Appl. Informatics*, vol. 4, no. 3, pp. 252–261, Dec. 2022, doi: 10.28926/ILKOMNIKA.V4I3.522.

[12] P. Bansal and A. Ouda, "Study on Integration of FastAPI and Machine Learning for Continuous Authentication of Behavioral Biometrics," *2022 Int. Symp. Networks, Comput. Commun. ISNCC 2022*, 2022, doi: 10.1109/ISNCC55209.2022.9851790.

[13] B. A. Baransel, A. Peker, H. O. Balkis, and I. Ari, "Towards Low Cost and Smart Load Testing as a Service Using Containers," *Commun. Comput. Inf. Sci.*, vol. 1382, pp. 292–302, 2021, doi: 10.1007/978-3-030-71711-7_24.

[14] L. Cardoso Silva *et al.*, "Benchmarking Machine Learning Solutions in Production," *Proc. - 19th IEEE Int. Conf. Mach. Learn. Appl. ICMLA 2020*, pp. 626–633, Dec. 2020, doi: 10.1109/ICMLA51294.2020.00104.

[15] N. Husufa and I. Prihandi, "Optimizing JMeter on Performance Testing Using the Bulk Data Method," *J. Inf. Syst. Informatics*, vol. 4, no. 2, pp. 205–215, Jun. 2022, doi: 10.51519/JOURNALISI.V4I2.244.

[16] E. N. Alam and F. Dewi, "PERFORMANCE TESTING ANALYSIS OF BANDUNGTANGINAS APPLICATION WITH JMETER," *Int. J. Innov. Enterp. Syst.*, vol. 6, no. 02, pp. 157–166, Jul. 2022, doi: 10.25124/IJIES.V6I02.172.

[17] R. Dhuny, A. A. I. Peer, N. A. Mohamudally, and N. Nissanke, "Performance evaluation of a portable single-board computer as a 3-tiered LAMP stack under 32-bit and 64-bit Operating Systems," *Array*, vol. 15, p. 100196, Sep. 2022, doi: 10.1016/J.ARRAY.2022.100196.