Jurnal Ilmu Komputer Agri-Informatika

Tersedia secara daring di:
https://jurnal.ipb.ac.id/index.php/jika

Volume 12 Nomor 2 halaman 142–154
eISSN: 2654-9735, pISSN: 2089-6026

Pengembangan Model LMS Berbasis *Serverless* untuk Mengatasi Masalah Kinerja di Lingkungan Padat Pengguna

Designing LMS Model Based on Serverless to Solve Performance Issue on High Concurrency Workload

MUHAMMAD JAKA UTAMA*, SHELVIE NIDYA NEYMAN, KARLISA PRIANDANA

Abstrak

Moodle adalah salah satu aplikasi *Learning Management System* (LMS) yang berperan penting dalam pembelajaran secara daring. Salah satu faktor penting yang perlu diperhatikan dari LMS Moodle adalah kinerja di lingkungan padat pengguna. Padat pengguna mengacu pada situasi di mana jumlah pengguna atau peserta dalam sistem melebihi kapasitas yang dapat ditangani oleh sistem tersebut. Penelitian ini bertujuan untuk menghasilkan suatu pilihan model arsitektur LMS Moodle agar dapat menangani masalah kinerja dalam lingkungan padat pengguna. Pendekatan yang digunakan adalah arsitektur *hybrid* yang menggabungkan teknologi *serverless* pada komponen *database*, penyimpanan data, dan *session handler*, serta *container* di atas *virtual machine* (VM) dengan layanan IaaS pada *core system* dengan dukungan *load balancing* dan *auto scaling*. Penelitian dilakukan melalui empat tahap, yaitu identifikasi masalah, perancangan, implementasi, dan analisis model LMS. Hasil evaluasi menunjukkan bahwa model LMS yang dikembangkan mampu menangani hingga 1500 pengguna bersamaan tanpa penurunan kinerja signifikan, dengan *response time* di bawah 2500 ms dan *failure request* di bawah 1%. Pengujian lanjutan dengan konfigurasi *batas minimum resource* memungkinkan sistem melayani hingga 10.000 pengguna secara simultan. Skor *benchmark plugin* Moodle menunjukkan performa optimal pada seluruh aspek. Model ini terbukti dapat meningkatkan kehandalan dan skalabilitas LMS di lingkungan padat pengguna.

Kata Kunci**:** Kinerja *serverless*, *learning management system*, Moodle, padat pengguna.

Abstract

Moodle is one of the Learning Management Systems (LMS) that play a critical role in supporting online education. One of the important factors in Moodle LMS is its performance in high concurrency workload environments. High concurrency workload refers to situations where the number of users exceeds the system's capacity. This study aims to propose an architectural model for Moodle LMS that can address performance issues under such heavy-load conditions. The proposed approach adopts a hybrid architecture that integrates serverless technologies for database, file storage, and session handler components, along with containerization on virtual machines (VM) using IaaS for the core system, supported by load balancing and autoscaling mechanisms. The research follows four stages: problem identification, system design, implementation, and LMS model analysis. Evaluation results show that the developed LMS model can handle up to 1500 concurrent users without significant performance degradation, maintaining a response time below 2500 ms and a failure request rate below 1%. Further testing with minimum resource configuration allows the system to support up to 10.000 concurrent users. Benchmark scores using the Moodle plugin indicate optimal performance across all aspects. This model has proven to enhance the reliability and scalability of LMS platforms in high-concurrency workload environments.

*Keywords**:** High concurrency workload, learning management system, Moodle, serverless.*

PENDAHULUAN

Learning Management System (LMS) adalah perangkat lunak yang dirancang untuk mengelola proses pembelajaran secara *online*. LMS sudah digunakan sejak lama khususnya perguruan tinggi untuk menyediakan lingkungan belajar *virtual* yang terintegrasi. LMS

Program Studi Ilmu Komputer, Sekolah Sains Data, Matematika, dan Informatika, Institut Pertanian Bogor, Bogor 16680;
*Penulis Korespondensi: Tel/Faks: 081330304032; Surel: mjakautama@apps.ipb.ac.id

memainkan peran penting dalam mendukung pendidikan dan pelatihan jarak jauh, pembelajaran berbasis *online* (Ahmed *et al*. 2021), dan mengelola konten serta interaksi pembelajaran secara efisien dalam lingkungan digital (Setiawan 2021). Contoh LMS yang populer adalah Moodle, Blackboard, Canvas, dan Google Classroom. Dalam perkembangannya, LMS dengan menggunakan Moodle paling umum digunakan di bidang pendidikan (Jingga dan Sunindyo 2020).

Salah satu faktor penting yang perlu diperhatikan dari LMS Moodle adalah kinerja di lingkungan padat pengguna (Sadikin *et al*. 2019; Mihai *et al*. 2023). Padat pengguna mengacu pada situasi di mana jumlah pengguna atau peserta dalam sistem melebihi kapasitas yang dapat ditangani oleh sistem tersebut. Dalam kondisi padat pengguna, beban kerja dan permintaan yang tinggi dapat mempengaruhi kinerja aplikasi, seperti waktu respons yang lambat, penurunan kecepatan akses, atau bahkan kegagalan sistem. Pada umumnya masalah padat pengguna merupakan suatu masalah *load balancing* agar beban tidak terpusat pada satu *server* (Mihăescu *et al*. 2011; Sadikin *et al*. 2019; Alier *et al*. 2020; Kacapor dan Veselinovic 2021; Zaini *et al*. 2021; Mihai *et al*. 2023). Penerapan *load balancing* dapat mengatasi masalah kinerja. Namun, karena tingginya jumlah pengguna, diperlukan daya komputasi yang semakin besar. Oleh karena itu, penggunaan *load balancing* saja tidak lagi memadai, sehingga diperlukan suatu pemilihan *deploy* LMS yang tepat untuk mengatasinya (Yang *et al*. 2017).

Moodle dapat di-*deploy self-hosted* (*on-prem*) atau di-*deploy* pada lingkungan *cloud* (*serverless*) (Mihai *et al*. 2023). Saat ini, penelitian terkait *deploy* suatu aplikasi menggunakan arsitektur *serverless* menjadi semakin populer. *Serverless* mengacu pada komputasi awan di mana teknisi tidak perlu memikirkan mengurus *server* dari *provision*, *maintenance*, *scaling*,dan *patching* karena semua dilakukan oleh penyedia *cloud* (Kumar 2024). Contoh penyedia *cloud* yang populer adalah Amazon Web Service (AWS), Microsoft Azure, dan Google Cloud. Hal ini terlihat dari hasil penelitian yang menggunakan arsitektur *serverless*, diantaranya kinerja (Zhang dan Zhu 2017), biaya (Abdulmohson *et al*. 2022), fleksibilitas dan skalabilitas (Al-Dhuraibi *et al*. 2017). Begitu pula untuk menangani fluktuatif pengguna LMS, penggunaan *serverless* dapat mengoptimalkan *resource* dalam penerapannya (Nday *et al*. 2023).

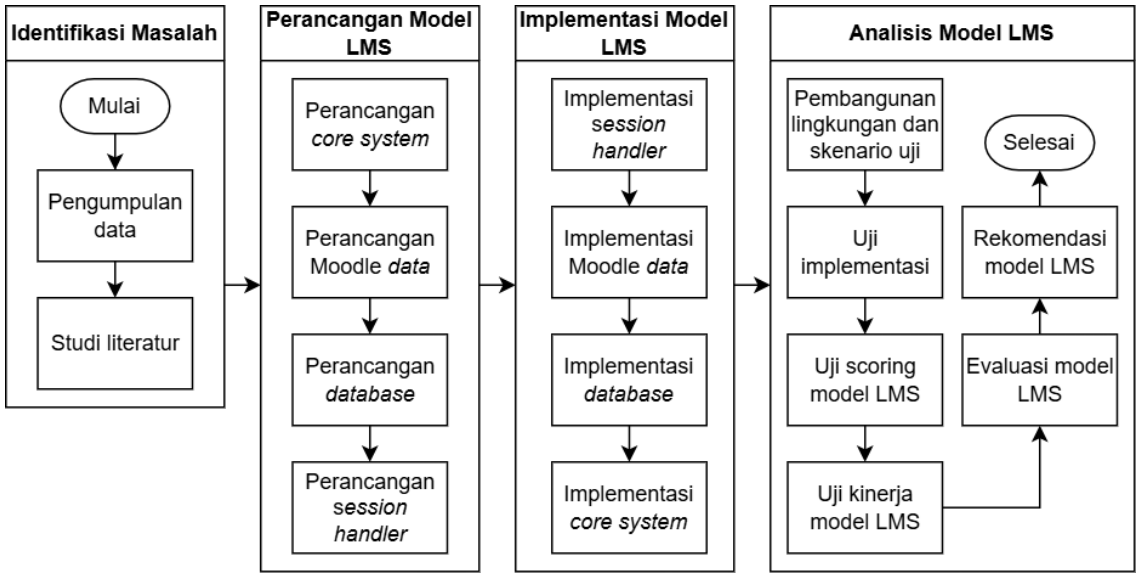
Sebagai langkah awal, kajian dilakukan pada LMS IPB *University*. IPB *University* merupakan salah satu universitas di Bogor yang didirikan sejak tahun 1963 dan menggunakan Moodle sebagai LMS. IPB *University* memiliki banyak program studi yang tersebar di berbagai fakultas dan memiliki sekitar kurang lebih 28.000 mahasiswa aktif setiap tahunnya. Hampir semua kegiatan akademis dilakukan di LMS, mulai dari penyebaran materi perkuliahan, *quiz*, ujian tengah semester (UTS), ujian akhir semester (UAS), hingga pengumpulan tugas dan projek mahasiswa. Infrastruktur dan aplikasi LMS IPB *University* dikelola oleh Lembaga Manajemen Informasi dan Transformasi Digital (LMITD) di *on-prem* IPB *University*. Meskipun sumber daya *server* dan teknis tersedia, serta telah digunakan selama bertahun-tahun di IPB *University*, transisi ke aktivitas *online* menyebabkan berbagai masalah yang harus diatasi salah satunya kinerja pada lingkungan padat pengguna. Pada lingkungan padat pengguna, LMS harus dapat menangani sejumlah besar sesi secara bersamaan pada jam-jam perkuliahan antara jam 08.00 sampai 17.00 dari Senin hingga Jumat. Selain itu, pada akhir pekan, jam istirahat, dan di luar jam kantor, LMS harus dapat menangani semua permintaan karena mahasiswa pada jam tersebut melakukan pengumpulan tugas dan dosen banyak memperbarui materi perkuliahan. LMS IPB *University* ini penting dalam menunjang perkuliahan. Kinerja LMS yang buruk, kecepatan akses yang lambat, dan ketidakstabilan sistem dapat mengganggu pengalaman belajar *online* (Jiwo dan Kusuma 2021), menghambat interaksi antara pengguna (Widiyono 2021), dan mempengaruhi efektivitas pembelajaran (Risyah 2022).

Penelitian ini bertujuan untuk menghasilkan suatu pilihan model arsitektur LMS Moodle agar dapat menangani masalah kinerja dalam lingkungan padat pengguna. Pengembangan model LMS Moodle pada penelitian ini menggunakan *arsitektur hybrid* berbasis *serverless* dengan menggabungkan teknik *load balancing*. *Serverless* untuk *database*, *storage* dan *session handler* menggunakan layanan PaaS. Aplikasi utama menggunakan layanan IaaS dengan

container Docker yang dijalankan diatas *virtual machine*. Layanan *serverless* menggunakan AWS karena pada penelitian (Wiechork dan Charão 2020) AWS memberikan kinerja yang lebih baik.

METODE

Penelitian ini dilakukan dalam empat tahapan, yaitu identifikasi masalah, perancangan model LMS, implementasi model LMS, dan analisis model LMS. Alur tahapan penelitian ini ditunjukkan pada Gambar 1.



Gambar 1 Tahapan-tahapan yang dilakukan pada penelitian

Identifikasi Masalah

Pada tahap ini dilakukan studi literatur untuk memperoleh informasi tambahan terkait faktor-faktor yang memengaruhi padat pengguna di LMS, yang akan memperkuat argumen pada tahap perancangan. Studi literatur dilakukan dengan pendekatan sistematis, menggunakan referensi publikasi seperti *conference*, *journal*, dan buku yang berfokus pada penelitian tentang LMS Moodle, *serverless*, dan teknik *load balancing* dalam beberapa tahun terakhir.

Pengambilan data juga dilakukan menggunakan data yang diambil pada pertengahan perkuliahan semester genap (Mei-Juli) 2023 dari laporan di grup LMS dan *help center* terkait *error* aplikasi. Data *analytics* IPB *University* diambil untuk melihat pengunjung ke aplikasi. Selain itu juga dilakukan survei terkait tempat *hosting* LMS. Tujuan pengambilan data untuk mengetahui contoh *real* keadaan padat pengguna dan menambah argumen dalam perancangan model LMS selain dari hasil studi literatur.

Perancangan Model LMS

Pada tahap ini, dilakukan perancangan arsitektur berdasarkan hasil identifikasi masalah, yaitu merancang model arsitektur *serverless*, termasuk menentukan layanan-layanan apa saja yang akan digunakan dalam *serverless*. Hasil dari perancangan ini adalah model LMS berbasis *serverless* yang dilanjutkan ke tahap implementasi.

Dalam perancangan LMS, karena kompleksitas aplikasinya, terdapat teknik-teknik khusus yang perlu diperhatikan. Dalam penelitian (Zaini *et al*. 2021) beberapa komponen penting yang harus diperhatikan dalam perancangan LMS menggunakan Moodle adalah:

a. *Core system* digunakan untuk menjalankan aplikasi Moodle. Di dalamnya terdiri dari *code-code* aplikasi Moodle, *theme*, dan *plugin*. Bagian ini secara langsung akan diakses oleh pengguna dan semua fungsi Moodle berada di sini.
b. Moodle *data* berisi *file-file* hasil unggahan pengguna dan *file* hasil proses *core system*, seperti *file* pdf, presentasi, *image*, video, *file backup*. Bagian ini harus berada di lokasi

terpisah dengan *core system* untuk menghindari unggahan *file malware* oleh pengguna tetapi harus tetap dapat diakses oleh *web server*.

c. *Database* berisi semua *database* berkaitan dengan Moodle *system data*.

Implementasi Model LMS

Pada tahap ini, model LMS hasil perancangan pada tahapan sebelumnya diimplementasi dalam *cloud*. Tahapan ini mencakup instalasi Moodle, penyesuaian (*tuning*) aplikasi, serta implementasi teknologi yang ada di *cloud*. Hasil dari tahap ini adalah LMS yang siap digunakan, yang dilanjutkan ke tahap analisis dan evaluasi.

Analisis Model LMS

Analisis dilakukan dalam tiga fase. Fase pertama analisis hasil implementasi model LMS. Fase kedua analisis hasil *scoring* model LMS. Fase ketiga analisis hasil pengujian kinerja model LMS. Analisis hasil implementasi model LMS bertujuan untuk melihat bagaimana model LMS saat digunakan secara langsung. Analisis ini dilakukan berdasarkan hasil data yang diambil pada semester ganjil (Agustus-Desember) 2023. Data diambil dari laporan di grup LMS dan *help center* terkait *error* aplikasi untuk melihat *improvement* dari model LMS. *Data analytics* IPB *University* diambil untuk melihat *trend* pengunjung ke aplikasi selama menggunakan model LMS yang dihasilkan.

Model LMS yang dibuat harus dilakukan penilaian dasar untuk mengetahui kualitas modelnya. Salah satu alat yang direkomendasikan Moodle untuk melakukan *scoring* model LMS adalah Moodle *benchmark plugin* (Jingga 2020). *Plugin* ini digunakan sebagai penilaian dasar kecepatan dan kualitas model LMS menilai dari semua aspek *critical* LMS yaitu kecepatan *server*, kecepatan prosesor, kecepatan *storage*, kecepatan *database,* dan kecepatan membuka suatu halaman. Hasilnya adalah *score* dari waktu total pada setiap aspek yang diperiksa dengan tiap-tiap aspek memiliki *acceptable limit* dan *critical limit*. Semakin kecil nilai yang didapatkan, maka kinerja model LMS semakin baik. Aspek yang diukur dan penjelasannya dapat dilihat pada Tabel 1.

Tabel 1 *Parameter* pengujian

#	*Description*	*Acceptable limit*	*Critical limit*
1	*Moodle loading time* *Load the "config.php" configuration file*	0.5	0.8
2	*Processor processing speed* *Call a PHP function with a loop to check the processor speed*	0.5	0.8
3	*Reading file performance* *Read a file multiple times to check the reading speed of the Moodle temporary folder*	0.5	0.8
4	*Writing file performance* *Write a file multiple times to check the writing speed of the Moodle temporary folder*	1	1.25
5	*Reading course performance* *Read a course multiple times to check the reading speed of the database*	0.75	1
6	*Writing course performance* *Write a course multiple times to check the writing speed of the database*	1	1.25
7	*Database performance* (#1) *Run a complex SQL query to check the speed of the database*	0.5	0.7
8	*Database performance* (#2) *Run a complex SQL query to check the speed of the database*	0.3	0.5
9	*Login time of administration notification page* *Load the administration interface notification page a few times to check web server speed*	0.3	0.8

Analisis hasil pengujian kinerja model LMS dilakukan dengan simulasi menggunakan *bot*. Sejumlah *bot* dibangkitkan untuk melakukan tugas *login*, melihat kursus, dan mencoba kuis secara bersamaan. Jumlah *bot* yang digunakan adalah 100, 500, 1500, 2500, 5000, 7500, dan 10.000, dengan waktu peningkatan bertahap (*ramp-up period*) selama 5 menit. Angka ini diambil dengan mempertimbangkan *bottleneck* dari sistem, sehingga digunakanlah pola

tersebut (Draheim *et al*. 2006). Pada tahap ini, kesimpulan diambil berdasarkan temuan penelitian, sekaligus menjawab pertanyaan penelitian yang diajukan. Tahap ini juga menghasilkan rekomendasi mengenai jumlah sumber daya optimal yang diperlukan, sesuai dengan target jumlah pengguna, untuk meningkatkan kinerja dan skalabilitas Moodle dalam lingkungan padat pengguna. Pengujian ini menggunakan *clone* LMS pada implementasi model LMS dengan mempertimbangkan *concurrent users* terhadap *response time* (ms) dan *failure request* (%) serta melihat pemakaian *resource* seperti CPU (*core*), *memory* (GB), *database* (ACU), dan *bandwidth* (MBps). Parameter yang digunakan pada pengujian ini termasuk *tools* yang digunakan untuk mengambil data dapat dilihat pada Tabel 2.

Tabel 2 *Parameter* pengujian

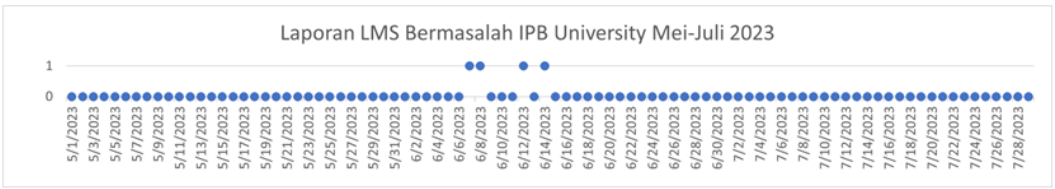
Komponen	**Nilai**	***Tools***
CPU ec2	*core*	AWS CloudWatch
Memory ec2	GB	AWS CloudWatch
Database	ACU	AWS CloudWatch
Bandwidth	MBps	AWS CloudWatch
Failure request	%	Locust
Response time	ms	Locust

HASIL DAN PEMBAHASAN

Identifikasi Masalah

Faktor penting yang harus dipertimbangkan untuk membuat model LMS padat pengguna adalah kemampuan sistem untuk dapat melayani beban tinggi dan fluktuatif (Nday *et al.* 2023). Hal ini dapat terlihat pada Gambar 2 yang menunjukkan jumlah pengunjung mengakses LMS IPB *University* pada pertengahan perkuliahan semester genap (Mei-Juli) 2023. Data diambil berdasarkan visitor dari aplikasi *analytics* IPB *University*. Terlihat jumlah pengunjung fluktuatif dan bisa mencapai lebih dari 16.000 dalam sehari.

Gambar 2 Jumlah pengunjung LMS IPB *University* setiap hari



Gambar 3 Jumlah pengunjung dan laporan *error* LMS IPB *University*

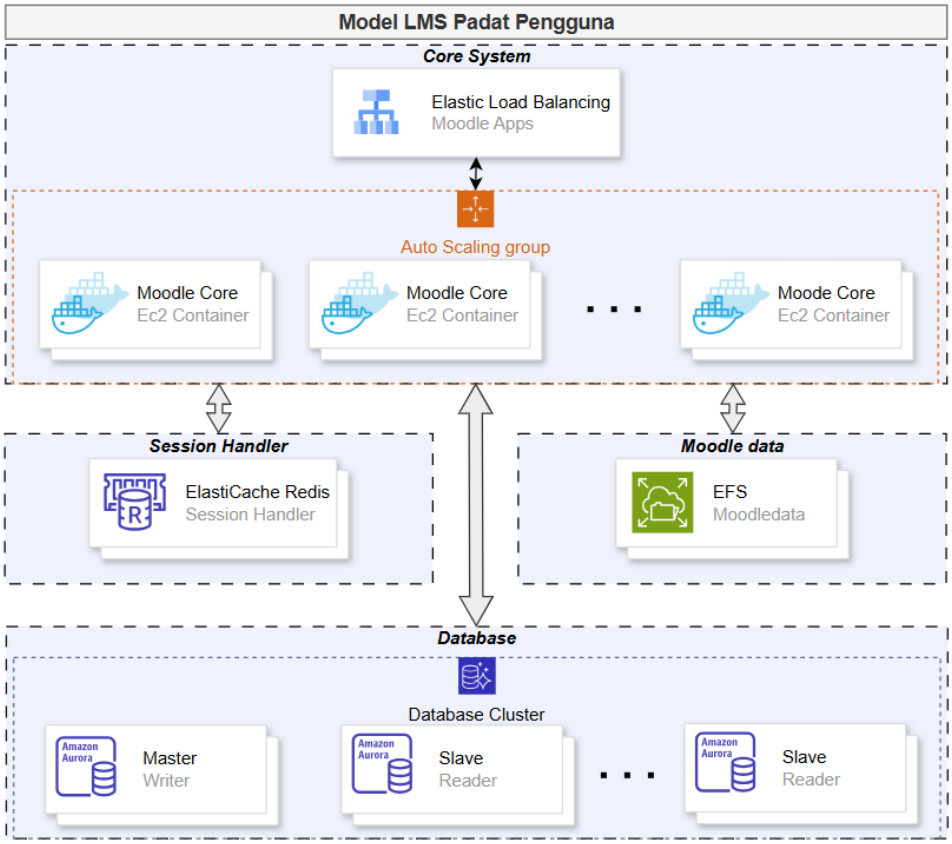
Pada Gambar 3 menunjukkan laporan mengenai terjadinya gangguan (*down*) pada aplikasi LMS. Data ini diambil dari laporan di grup LMS dan *help center*. Berdasarkan laporan *error* dan jumlah pengunjung yang mengakses, terlihat terjadi gangguan pada sistem ketika jumlah pengunjung lebih dari 9500 pengunjung per hari. Dari segi teknis, infrastruktur, dan aplikasi LMS IPB saat ini berjalan di *server on-prem* milik IPB dan memiliki keterbatasan sumber daya.

Berdasarkan hal tersebut, penelitian ini membuat model solusi LMS untuk lingkungan padat pengguna berbasis *serverless*. Pengembangan model LMS Moodle pada penelitian ini menggunakan arsitektur *hybrid* berbasis *serverless* dengan menggabungkan teknik *load*

balancing. *Serverless* untuk *database*, *storage,* dan *session handler* menggunakan layanan PaaS. Aplikasi utama menggunakan layanan IaaS dengan *container* Docker yang dijalankan di atas *virtual machine*.

Perancangan Model LMS

Dalam penelitian (Zaini *et al.* 2021) tedapat tiga komponen penting yang harus diperhatikan pada perancangan yaitu *core system*, Moodle *data*, dan *database*. Pada kasus padat pengguna ada satu komponen lagi yang harus diperhatikan yaitu *session handler* (Büchner 2022). *Session handler* digunakan untuk mengatur *session* pengguna.



Gambar 4 Model LMS

Perancangan Arsitektur LMS dimulai dari pemilihan tempat berjalannya *core system*. *Core system* digunakan untuk aplikasi Moodle berjalan, di dalamnya terdiri dari kode-kode aplikasi Moodle. Pada perancangan *core system* terdapat *load balancing* menggunakan ELB bertugas untuk mendistribusikan *traffic* ke beberapa *container* Moodle *core* yang berjalan di EC2. Cara *load balancing* ELB adalah membagi beban secara merata untuk meningkatkan *availability* dan *scalability*. ASG mengelola jumlah *instance* EC2 pada penelitian ini jika pemakaian CPU di atas 80% maka akan secara otomatis menambah *instance* EC2.

Moodle *data* digunakan untuk menyimpan *file* unggahan Moodle. Pada bagian ini diterapkan menggunakan teknologi EFS, yang akan di-*attach* pada *core system*. Moodle *data* dibuat terpisah karena memiliki *behaviour* dinamis, yang isinya berubah terus menerus. Berbeda dengan *core system* yang *static* serta hampir tidak ada perubahan.

Perancangan berikutnya adalah pada bagian *database* LMS. Pada perancangan *database* dibuat dalam *cluster* untuk pembagian beban kerja. Komponen *database* pada penelitian ini terdiri dari satu sebagai *master* yang difungsikan untuk melakukan semua proses *write* dari LMS dan *slave* yang bertindak untuk menangani semua permintaan *read only* dari LMS. Pada *slave* dibuat pengaturan, apabila pemakaian ACU di atas 80% maka akan menambahkan satu mesin *slave* lagi. Pembagian baca dan tulis pada bagian *database* bertujuan untuk meningkatkan kinerja model LMS.

Terakhir bagian *session handler*, bagian ini digunakan untuk menyimpan *session* peserta. Pada bagian ini diterapkan dengan *elasticache* berbasis Redis. Hal ini dipilih karena *behavior session* Moodle yang cepat dan harus *persistence*. Gambar 4 mengilustrasikan model LMS pada penelitian ini.

Implementasi Model LMS

Implementasi model LMS padat pengguna berbasis *serverless* ini diterapkan di AWS *region Southeast Asia* negara Singapura karena *service* yang tersedia lebih lengkap dibandingkan di negara Indonesia. Walaupun *server* di Singapura tetapi tidak mempengaruhi layanan karena berada satu *region* dengan Indonesia. Pada tahap implementasi dibagi ke dalam empat tahapan seperti pada Gambar 5.

Gambar 5 Urutan implementasi model LMS pada penelitian

1. *Session Handler*

Pada bagian *session handler* diterapkan dengan Amazon *elasticache serverless*, sehingga *resource* menyesuaikan dengan pemakaian. *Resource* yang disesuaikan yaitu *data* yang disimpan dan *Elasticache Processing Units* (ECPUs). Data yang disimpan dihitung dalam GB, untuk satu ECPUs setara dengan satu kali proses baca dan tulis dari setiap *kilobyte* (KB) data yang dikirim. Secara *default,* batas rendah *scaling elasticache* 1 GB sampai batas tertinggi 5000 GB *storage* dan 1000 ECPUs sampai batas tertinggi 15.000.000 ECPUs *request*. Pada penelitian ini batas yang diterapkan menggunakan *default elasticache* karena dengan *default* saja sudah cukup.

2. Moodle *data*

Pada bagian Moodle *data* diterapkan dengan EFS. Penggunaan ini karena fungsi Moodle *data* yang harus selalu sama isinya karena berkaitan dengan data aplikasi. EFS ini berbasis *serverless* mengikuti ukuran *storage* yang digunakan dan dapat mengembang hingga *exabytes* data. Sehingga pada penelitian ini tidak diatur ukuran *storage* dan otomatis mengikuti ukuran *data*.

3. *Database*

Pada implementasi *database cluster* dibuat menggunakan *service* Aurora *serverless* berbasis MySql. Mesin *database* ini dibuat dalam *cluster* dengan satu bertindak sebagai *master* dan satu lagi bertindak sebagai *slave*. *Database* yang digunakan menggunakan MySql. *Scaling* yang digunakan apabila rata-rata CPU sudah mencapai 80% maka akan membuat *instance slave database* baru. Satuan *resource* yang digunakan pada *database* ini adalah *Aurora Capacity Unit* (ACU). Setiap ACU setara dengan 1 *core* CPU dan 2 GB *memory*. Maksimum 1 *instance* Aurora bisa diatur *minimum* 0.5 ACU sampai 128 ACU. Pada penelitian ini digunakan *minimum* 0.5 ACU sampai 128 ACU.

4. *Core System*

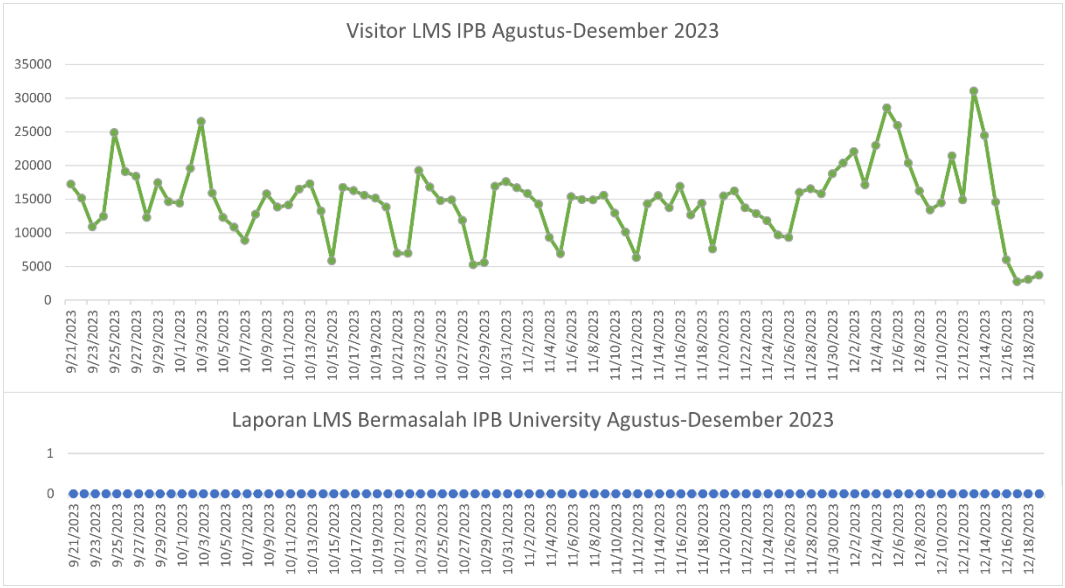
Bagian ini dibuat terakhir karena bagian inti yang memerlukan ketiga *requirement* terpenuhi. Pertama dibuat *template image* EC2 untuk dimasukkan ke dalam *autoscale*. Dalam *image* EC2 ini diinstal Moodle dan dibuat kontainer dengan basis *image* bitnami/moodle:4.1 yang dikostumisasi agar bisa berjalan di *environment serverless*. *Image* EC2 dilakukan konfigurasi seperti pengaturan *ulimit* dan *fstab* untuk *mounting* efs setiap *core system scaling*. Setelah diinstal dan dikonfigurasi lalu dibuat *template image* EC2. *Image* EC2 Moodle tersebut kemudian dimasukkan ke dalam *policy auto scaling*. Mesin yang digunakan c5a.xlarge karena memerlukan komputasi dan *bandwidth* yang besar dengan spesifikasi 4 *core* dan 8 GB *memory*. Batas *autoscaling* dibuat minimum 1 mesin dan akan bertambah mesin baru jika sudah menyentuh 80% penggunaan CPU. Nilai 80% ini digunakan karena ketika penambahan mesin dibutuhkan waktu untuk mesin barunya dapat digunakan. Dengan diberikan 80% mesin yang

available sekarang tidak penuh dahulu hingga mesin baru dapat digunakan. Setelah *autoscaling policy* jadi, lalu dimasukkan ke dalam *elastic load balancing* untuk membagi beban.

Analisis Model LMS

1. Hasil implementasi model LMS

Model LMS sudah diterapkan dan dipakai sehari-hari pada semester ganjil (Agustus-Desember) 2023. Pada Gambar 6 menunjukkan terjadi peningkatan data pengunjung yang mengakses dengan model arsitektur LMS. Data tersebut diambil dari laporan di grup LMS dan *help center* terkait *error* aplikasi untuk melihat *improvement* dari model LMS serta data *analytics* IPB *University* untuk melihat *trend* pengunjung ke aplikasi selama menggunakan model LMS yang dihasilkan. Selama implementasi model arsitektur LMS tidak terjadi aplikasi LMS *down* ataupun keluhan dari pengguna. Jumlah pengunjung LMS terjadi peningkatan setelah implementasi terjadi kenaikan hingga lebih dari 30.000 ketika *peak* dalam sehari. Model LMS pada penelitian ini mampu menangani peningkatan jumlah pengunjung.



Gambar 6 Jumlah pengunjung dengan laporan LMS hasil implementasi

2. Hasil *scoring* model LMS

Pada tahap ini dilakukan penilaian kinerja dasar model LMS, dengan menggunakan Moodle *benchmark plugin*. *Plugin* ini mengecek semua aspek *critical* LMS yaitu kecepatan *server*, kecepatan prosesor, kecepatan *storage*, kecepatan *database,* dan kecepatan membuka suatu halaman, yang mana jika salah satu aspek mendapatkan *score* jelek akan mempengaruhi sistem LMS keseluruhan. Semakin kecil *score* yang dihasilkan dan semua *score* berwarna hijau mengindikasikan bahwa situs LMS tersebut baik. Hasil *scoring* model LMS dapat dilihat pada Tabel 3.

Hasil Moodle *loading time* untuk mengetahui seberapa cepat model LMS dapat membaca *file* konfigurasi didapatkan nilai 0.003 detik, menunjukkan nilai yang baik karena berada jauh di bawah batas yang dapat diterima yaitu 0.5 detik dan batas kritis 0.8 detik. Hasil *processor processing speed* untuk mengetahui suatu model LMS memproses fungsi aplikasi berulang tanpa hambatan didapatkan nilai 0.06 detik, menunjukkan kinerja prosesor baik karena berada jauh di bawah batas yang dapat diterima yaitu 0.5 detik dan batas kritis 0.8 detik. Hasil *reading file performance* untuk mengetahui kecepatan membaca *file* di direktori model LMS didapatkan nilai 0.016 detik, menunjukkan kecepatan membaca optimal karena jauh di bawah batas yang dapat diterima yaitu 0.5 detik dan batas kritis 0.8 detik. Hasil *writing file performance* untuk mengetahui kecepatan menulis *file* di direktori model LMS didapatkan nilai 0.077 detik, menunjukkan kecepatan menulis baik karena hanya 7.7% dari batas maksimal yang dapat diterima yaitu 1 detik. Hasil *reading course performance* untuk mengetahui kinerja membaca

course pada *database* model LMS didapatkan nilai 0.078 detik, menunjukkan kinerja membaca baik karena jauh di bawah batas yang dapat diterima yaitu 0.75 detik. Hasil *writing course performance* untuk mengetahui kinerja menulis *course* pada *database* model LMS didapatkan nilai 0.026 detik, menunjukkan kinerja menulis yang baik karena jauh di bawah batas yang dapat diterima yaitu 1 detik. Hasil *database performance* (#1) untuk melihat kinerja *database* dalam menjalankan *query* SQL kompleks didapatkan nilai 0.058 detik, menunjukkan kinerja *database* baik karena jauh di bawah batas yang dapat diterima yaitu 0.5 detik. Hasil *database performance* (#2) untuk melihat kinerja menjalankan SQL selain *query* kompleks didapatkan nilai 0.081 detik, menunjukkan kinerja *database* baik karena jauh di bawah batas yang dapat diterima yaitu 0.5 detik. Hasil *loading time of administration notification page* untuk melihat waktu *loading* halaman admin didapatkan nilai 0.063 detik, menunjukkan waktu *loading* cepat karena jauh di bawah batas yang dapat diterima yaitu 0.8 detik.

Total waktu keseluruhan 0.462 atau 47 *points* menunjukkan model LMS berada dalam kategori optimal. Semua pengujian berada jauh di bawah batas yang dapat diterima maupun kritis. Berdasarkan hasil tersebut model LMS ini tidak perlu peningkatan atau revisi pada elemen-elemen yang diuji.

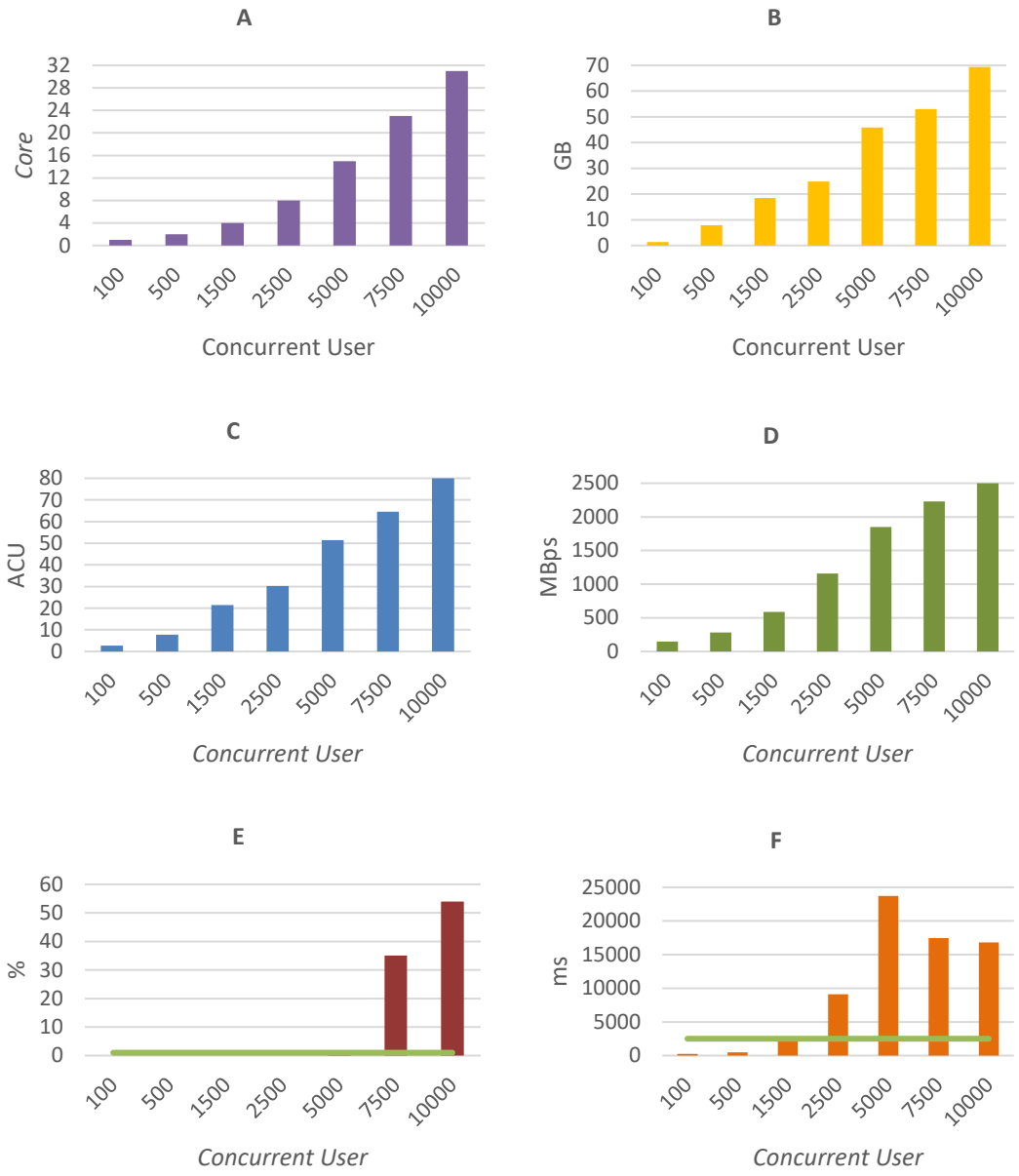
Tabel 3 Hasil *scoring* model LMS

#	*Description*	*Time* (*seconds*)
1	*Moodle loading time*	0.003
2	*Processor processing speed*	0.060
3	*Reading file performance*	0.016
4	*Writing file performance*	0.077
5	*Reading course performance*	0.078
6	*Writing course performance*	0.026
7	*Database performance* (#1)	0.058
8	*Database performance* (#2)	0.81
9	*Login time of administration notification page*	0.063
	Total time	0.462
	Score	47 *points*

3. Hasil pengujian kinerja model LMS

Pengujian simulasi dengan *bot* ini menggunakan *locust* dengan membangkitkan sejumlah *bot* untuk melakukan *task login*, *view course*, *attempt quiz*, menjawab soal, dan *submit quiz* secara bersamaan dengan jumlah *concurrent user* 100, 500, 1500, 2500, 5000, 7500 dan 10,000. *Output* hasil pengujian model arsitektur LMS menggunakan *locust* ini adalah *response time* serta jumlah *failure request*. *Response time* adalah waktu yang dibutuhkan dari pengguna meminta *request* hingga kembali lagi ke pengguna. *Failure request* adalah jumlah pengguna yang tidak dapat dilayani. Keadaan *resource* juga dipantau selama pengujian menggunakan *locust*. *Resource* yang dipantau adalah CPU, *memory*, ACU, dan *bandwidth*. *Data resource* selama pengujian didapatkan dengan menggunakan AWS *CloudWatch*. Pada pengujian simulasi pertama, konfigurasi *autoscale* batas minimum mesin EC2 dan *database* tidak diatur. Hasil pengujian dapat dilihat pada Gambar 7.

Pada CPU EC2, penggunaan CPU meningkat secara signifikan ketika jumlah *concurrent user* bertambah. Kenaikan terlihat eksponensial, terutama mulai dari 2500 hingga 10.000 *concurrent user*. Hal ini menunjukkan bahwa kebutuhan CPU meningkat tajam terhadap jumlah *concurrent user*. Pada *memory* EC2, penggunaan *memory* juga meningkat seiring bertambahnya jumlah *concurrent user*. Pola kenaikan serupa dengan CPU, dengan lonjakan besar mulai dari 2500 *concurrent user*, menunjukkan bahwa pemakaian *memory* memiliki ketergantungan langsung terhadap jumlah pengguna. Pada *database* aktivitas diukur dalam ACU, penggunaan ACU meningkat linier hingga *concurrent user* 10.000. Hal ini menunjukkan bahwa beban pada *database* semakin besar dengan bertambahnya *concurrent user*. Pada *bandwidth*, penggunaan *bandwidth* meningkat dengan jumlah *concurrent user*. Kenaikan mulai signifikan dari 2500 pengguna, dengan puncak pada 10.000 pengguna yang memerlukan *bandwidth* lebih dari 2500 MBps.



Gambar 7 Hasil pengujian simulasi pertama. (a) CPU EC2; (b) Memori EC2; (c) *Database*; (d) *Bandwidth*; (e) *Failure Request*; (f) *Response Time*

Pada *failure request* terjadi pola kenaikan ketika jumlah *concurrent user* 100 hingga 2500 tidak terjadi *failure request*. Pada 5000 *concurrent user* mulai terjadi 1%, 7500 terjadi 35%, 10.000 terjadi 54%. Hal ini menunjukkan dengan model LMS *serverless* tanpa diatur batas minimum persentase *failure request* mulai muncul pada 5000 *concurrent user* dan meningkat tajam pada 7500 *concurrent user*.

Pada *response time*, waktu respons meningkat drastis ketika jumlah *concurrent user* 2500 hingga 10.000. Pada jumlah *concurrent user* 7500 dan 10.000 waktu respons lebih rendah daripada 5000 dikarenakan persentase *failure request* lebih banyak.

Pada hasil pengujian simulai pertama pada Gambar 7 dapat dikatakan bahwa model LMS ini dapat dihandalkan hingga 5000 *concurrent user* berdasarkan *failure request* dan hingga 1500 *concurrent user* berdasarkan *response time*. Nilai maksimum *response time* harus di bawah 2500 ms untuk dikatakan baik (Al-Nuaimi *et al*. 2022) dan *failure request* harus di bawah 1% (Burgess 2016), sehingga dilakukan percobaan kedua. Pada percobaan kedua dilakukan pengoptimalan *autoscale*, perubahan batas minimum jumlah mesin EC2 pada *load balancing*, dan ACU *database*. Nilai batas minimum diambil berdasarkan hasil pengujian simulasi pertama.

Pada simulasi pertama jumlah CPU, *memory*, *database,* dan *bandwidth* pola pemakaiannya terlihat linear terhadap jumlah *concurrent user* sehingga dapat digunakan pendekatan regresi linear sederhana. Persamaan 1 merupakan persamaan yang mengilustrasikan penentuan nilai batas minimum jumlah CPU, *memory*, *database*, dan *bandwidth* untuk pengujian kedua dengan persamaan linear.

$$y = a \cdot x + b \tag{1}$$

dengan y nilai *resource* seperti CPU, *memory*, *database,* dan *bandwidth* yang akan dicari. x pada penelitian ini adalah jumlah *concurrent user*. a adalah koefisien kemiringan (*slope*). Pada penelitian ini a menunjukkan tingkat kenaikan kebutuhan sumber daya (CPU, *memory*, *database*, *bandwidth*) bersamaan dengan *concurrent user*. b adalah *intercept* (titik potong). Pada penelitian ini b menunjukkan kebutuhan *resource* minimum yang diperlukan bahkan jika tidak ada *concurrent user*. Rumus penentuan nilai batas minimum CPU, *memory*, *database,* dan *bandwidth* pada model LMS penelitian ini dapat dihitung dengan Persamaan 2, 3, 4 dan 5.

$$CPU = 0{,}003 \times Concurrent\ Users + 1{,}4 \tag{2}$$

$$Memory = 0{,}0064 \times Concurrent\ Users + 2 \tag{3}$$

$$Database = 0{,}008 \times Concurrent\ Users + 3 \tag{4}$$

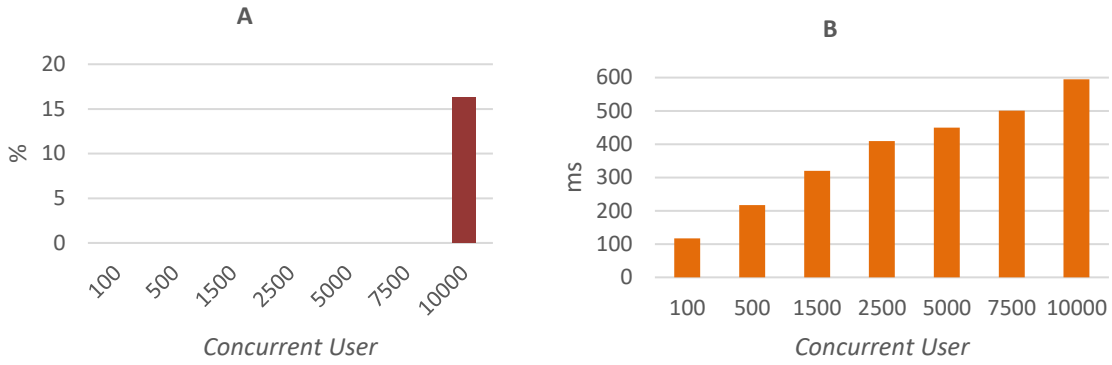
$$Bandwidth = 0{,}24 \times Concurrent\ Users + 100 \tag{5}$$

Dari persamaan tersebut kemudian dilakukan perhitungan nilai batas minimum. Nilai batas minimum *resource* dijadikan dasar untuk dilakukan pada pengujian kedua yang dapat dilihat pada Tabel 4. Nilai untuk CPU dan *memory* dibulatkan ke atas dikarenakan tidak ada bentuk koma untuk CPU dan *memory*. Bagian *bandwidth* pada penelitian ini tidak bisa ditambah lagi karena *limit* pada akun AWS yang digunakan maksimal hanya 2500 MBps.

Tabel 4 Nilai minimum setiap komponen terhadap jumlah *concurrent user*

Concurrent User	*Core System*		*Database* (ACU)	*Bandwidth* (MBps)
	CPU ec2 (Core)	Memory ec2 (GB)		
100	2	3	3,8	124
500	3	6	7	220
1.500	6	12	15	460
2.500	9	18	23	700
5.000	17	34	43	1300
7.500	24	50	63	1900
10.000	32	66	83	2500

Pengujian simulasi kedua sama seperti yang pertama tetapi hanya menampilkan bagian *failure request* dan *response time* dikarenakan fokusnya untuk melihat perbaikan dari kedua hasil tersebut. Pada Gambar 8 didapatkan nilai yang baik dikarenakan semua nilai *respons time* berada di bawah 2.5 s. Akan tetapi pada jumlah *concurrent* 10.000 masih ada yang *error* dikarenakan limitasi jumlah *bandwidth* maksimal 2500 MBps pada akun AWS yang digunakan sehingga ada beberapa *user* yang tidak mampu dilayani secara bersamaan. Tetapi pada *response time,* model LMS ini masih baik sehingga dapat dibuktikan bahwa model LMS pada penelitian ini mampu mengatasi masalah padat pengguna.



Gambar 8 Hasil pengujian simulasi kedua. (a) *Failure Request*; (b) *Response Time*

SIMPULAN

Penelitian ini berhasil menghasilkan model LMS Moodle berbasis *serverless* yang mampu mengatasi masalah kinerja pada lingkungan padat pengguna. Model yang dikembangkan memanfaatkan layanan *cloud* AWS, menggabungkan pendekatan *load-balancing*, *autoscale*, dan teknologi *container*. Komponen-komponen penting seperti *core system*, *database*, *session handler*, dan *file storage* dirancang agar dapat diskalakan secara otomatis menyesuaikan beban pengguna. Hasil evaluasi menunjukkan bahwa model LMS dapat menangani hingga 1500 *concurrent users* tanpa mengalami penurunan kinerja signifikan, dengan *failure request* kurang dari 1% dan *response time* kurang dari 2500 ms tanpa ditentukan batas minimal *resource*. Jika target *concurrent user* lebih dari 1500, perlu ditentukan batas minimal *resource* dengan perhitungan untuk *core system* dan *database*. Skor *benchmark plugin* Moodle menunjukkan seluruh aspek berada di bawah *acceptable limit*, menandakan kinerja model LMS optimal. Pada evaluasi implementasi di IPB *University*, tidak ditemukan keluhan dan terjadi peningkatan jumlah pengunjung harian hingga lebih dari 30.000. Model LMS ini terbukti meningkatkan kinerja dan keandalan sistem dalam menghadapi lonjakan pengguna, menjadikan model LMS ini solusi untuk institusi pendidikan dengan beban pengguna tinggi.

DAFTAR PUSTAKA

Abdulmohson A, Kadhim, MF, Anssari OMH, Al-Jobouri AA. 2022. Cost analysis of on-premise versus cloud-based implementation. *Indonesian Journal of Electrical Engineering and Computer Science.* 25(3):1787-1794. doi:10.11591/ijeecs.v25.i3.pp1787-1794.

Ahmed W, Parveen Q, Dahar MA. 2021. Role of Learning Management System in Distance Education: A Case Study of Virtual. *Sir Syed Journal of Education & Social Research.* 4(1):119-125. doi:10.36902/sjesr-vol4-iss1-2021.

Al-Dhuraibi Y, Paraiso F, Djarallah N, Merle P. 2017. Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Transactions on Services Computing.* 11(2):430-447. doi:10.1109/TSC.2017.2711009.

Alier M, Casany MJ, Llorens A, Alcober J, Prat JD. 2020. Atenea Exams, an IMS LTI Application to Solve Scalability Problems: A Study Case. *Applied Sciences.* 11(1):80. doi:10.3390/app11010080.

Al-Nuaimi MN, Sawafi OSA, Malik SI, Al-Emran M, Selim YF. 2022. Evaluating the Actual Use of Learning Management Systems During the Covid-19 Pandemic: an Integrated Theoretical Model. *Interactive Learning Environments.* 31(10):6905-6930. doi:10.1080/10494820.2022.2055577.

Büchner A. 2022. *Moodle 4 Administration: an Administrator's Guide to Configuring, Securing, Customizing, and Extending Moodle, Fourth Edition.* Packt Publishing.

Burgess M. 2016. *Google SRE - Site Reliability Engineering.* Oslo: O'Reilly Media, Inc.

Draheim D, Grundy J, Hosking J, Lutteroth, C, Weber G. 2006. Realistic load testing of web applications. *Conference on Software Maintenance and Reengineering (CSMR'06)*. Bari: IEEE. doi:10.1109/CSMR.2006.43.

Jingga K. 2020. Pembangunan Berbasis Komponen Menggunakan Moodle Sebagai Alternatif Pengembangan Perangkat Lunak E-Learning Studi Kasus: Sistem Manajemen Pengetahuan Andalalin [tesis]. Bandung[ID]: Institut Teknologi Bandung.

Jingga K, Sunindyo WD. 2020. Component-based development using moodle as alternative for e-learning software development. *12th International Conference on Information Technology and Electrical Engineering (ICITEE)*. Yogyakarta: IEEE. hlm 125-130. doi:10.1109/ICITEE49829.2020.9271670.

Jiwo DS, Kusuma WA. 2021. Penggunaan Moodle LMS UMM dalam Pembelajaran Jarak Jauh di Masa Pandemi. *Jurnal Syntax Admiration.* 2(9):1653:1662. doi: 10.46799/jsa.v2i9.310.

Kacapor K, Veselinovic T. 2021. Designing an adaptable high-availability E-learning framework using free and opensource technology. *15th International Technology, Education and Development Conference 2021; 2021 Mar 8-9. IATED*. hlm 8223-8232. doi:10.21125/inted.2021.1671.

Kumar S. 2024. Optimizing Resources in Serverless Architectures: A Comprehensive Review A. *TechRxiv.* doi:10.36227/techrxiv.172504025.57438488/v1.

Mihăescu MC, Burdescu DD, Mocanu M, Ionascu CM. 2011. Load balancing procedure for building distributed e-learning systems. *The Third International Conference on Mobile, Hybrid, and On-line Learning.* Guadeloupe: IARIA. hlm 82-87.

Mihai D, Mihailescu ME, Carabas M, Tapus N. 2023. Integrated High-Workload Services for E-Learning. *IEEE Access.* 11:8441-8454. doi:10.1109/ACCESS.2023.3238967.

Nday BA, Kusuma GP, Fredyan R. 2023. Serverless Utilization in Microservice E-Learning Platform. *Procedia Computer Science.* 216:204-212. doi:10.1016/j.procs.2022.12.128.

Risyah MM. 2022. Faktor-Faktor Keberhasilan Penggunaan Learning Management System (LMS) "BeSmart Elearning" Universitas Negeri Yogyakarta [tesis]. Yogyakarta: Universitas Gadjah Mada.

Sadikin M, Yusuf R, Rifai A. 2019. Load Balancing Clustering on Moodle LMS to Overcome Performance Issue of e-learning System. *Telkomnika.* 17(1):131-138. doi:10.12928/telkomnika.v17i1.10284.

Setiawan A. 2021. Implementasi Pembelajaran Jarak Jauh Menggunakan LMS. *Jurnal Bestari.* 2(1):1-22.

Widiyono A. 2021. Pengaruh Penggunaan LMS dan Aplikasi Telegram terhadap Aktivitas Belajar. *Jurnal Penelitian Ilmu Pendidikan.* 14(1):91-101.

Wiechork K, Charão AS. 2020. Investigating the performance of moodle database queries in cloud environments. *ICEIS 2020 - 22nd International Conference on Enterprise Information Systems.* SciTePress. hlm 269-275.

Yang CT, Yeh WT, Shih WC. 2017. Implementation and Evaluation of an e-Learning Architecture on Cloud Environments. *International Journal of Information and Education Technology.* 7(8).

Zaini A, Santoso H, Sulistyanto MT. 2021. Fault Tolerance Strategy to Increase Moodle Service Reliability. *Journal of Physics: Conference Series.* 1869(1):012095.

Zhang W, Zhu Y. 2017. A New E-learning Model Based on Elastic Cloud Computing for Distance Education. *Journal of Mathematics, Science and Technology Education.* 13(12): 8393-8403. doi:10.12973/ejmste/80800.