Jutisi: Jurnal Ilmiah Teknik Informatika dan Sistem Informasi
https://ojs.stmik-banjarbaru.ac.id/index.php/jutisi/index
Jl. Ahmad Yani, K.M. 33,5 - Kampus STMIK Banjarbaru
Loktabat – Banjarbaru (Tlp. 0511 4782881), e-mail: puslit.stmikbjb@gmail.com
e-ISSN: 2685-0893

Implementasi dan Optimasi Server JupyterHub sebagai Pendukung Pembelajaran di Laboratorium Informatika Universitas Janabadra

DOI: http://dx.doi.org/10.35889/jutisi.v15i3.3453

Sri Rahayu[1*], Muhammad Fuad[2]
Informatika, Universitas Janabadra, Yogyakarta, Indonesia
*e-mail *Corresponding Author*: ayu.dj@janabadra.ac.id

Abstract

Python programming laboratories often encounter challenges related to inconsistent software configurations, library installation errors, and the lack of centralized management of the practical learning environment. This study aims to design and implement JupyterHub as a centralized Python laboratory environment based on a local server at the Informatics Laboratory of Universitas Janabadra. The study employed an applied experimental approach consisting of system requirements analysis, architecture design, service implementation, and multi-user performance testing and evaluation. The results demonstrate that the proposed system was successfully implemented according to the designed architecture. JupyterHub and the Dask cluster operated in an integrated manner, while user authentication, automated user management, and resource optimization through the Idle Culler mechanism functioned effectively. Furthermore, monitoring using the Dask Dashboard indicated that computational tasks were successfully distributed across the cluster with stable server resource utilization. These findings indicate that the proposed system provides a reliable, centralized, and efficient environment for supporting Python programming laboratories in higher education.
***Keywords**: JupyterHub; Laboratory; Python; Practical Learning; Centralized Server*

Abstrak

Pelaksanaan praktikum pemrograman Python di laboratorium komputer sering menghadapi permasalahan perbedaan konfigurasi perangkat, kesalahan instalasi pustaka, serta keterbatasan pengelolaan lingkungan praktikum secara terpusat. Penelitian ini bertujuan merancang dan mengimplementasikan JupyterHub sebagai lingkungan praktikum Python terpusat berbasis server lokal di Laboratorium Informatika Universitas Janabadra. Pendekatan yang digunakan adalah eksperimen terapan melalui tahapan analisis kebutuhan, perancangan arsitektur sistem, implementasi layanan, serta pengujian dan evaluasi performa multiuser. Hasil penelitian menunjukkan bahwa sistem berhasil diimplementasikan sesuai rancangan, layanan JupyterHub dan *cluster* Dask berfungsi secara terintegrasi, serta mekanisme autentikasi, otomatisasi manajemen pengguna, dan optimasi sumber daya melalui *Idle culler* berjalan dengan baik. Monitoring menggunakan *Dashboard* Dask menunjukkan bahwa distribusi komputasi berlangsung secara normal dengan penggunaan sumber daya server yang stabil.
Kata kunci: *JupyterHub; Laboratorium; Python; Praktikum; Server Terpusat*

1. Pendahuluan

Perkembangan pembelajaran berbasis komputasi (*computational learning*) di perguruan tinggi telah meningkatkan kebutuhan akan lingkungan praktikum yang terstandarisasi, mudah diakses, dan mampu mendukung pembelajaran berbasis Python secara kolaboratif. Dalam konteks ini, bahasa pemrograman Python telah menjadi standar de facto dalam pendidikan komputasi modern karena sifatnya yang sederhana, ekosistem pustaka yang luas, serta

dukungan kuat terhadap berbagai domain komputasi ilmiah [1]. Oleh karena itu, efektivitas penyelenggaraan praktikum Python tidak hanya bergantung pada kurikulum, tetapi juga pada kesiapan infrastruktur yang mampu menjamin konsistensi lingkungan eksekusi, kemudahan akses, serta efisiensi pengelolaan oleh dosen maupun laboran [2], [3].

Di lingkungan Laboratorium Informatika Universitas Janabadra, pelaksanaan praktikum Python masih menghadapi sejumlah kendala teknis yang berdampak langsung terhadap kualitas pembelajaran. Variasi konfigurasi perangkat lunak pada komputer laboratorium sering menyebabkan ketidaksesuaian dependensi pustaka, error saat eksekusi program, serta waktu instalasi yang cukup panjang pada awal praktikum. Permasalahan ini semakin kompleks dengan adanya penggunaan perangkat pribadi mahasiswa yang memiliki spesifikasi dan sistem operasi berbeda-beda, sehingga menghasilkan inkonsistensi output program. Selain itu, dosen mengalami kesulitan dalam melakukan monitoring dan evaluasi secara seragam karena tidak adanya lingkungan eksekusi yang terpusat. Padahal, laboratorium telah memiliki sumber daya server yang potensial, namun belum dimanfaatkan secara optimal untuk mendukung pembelajaran berbasis komputasi terpusat. Permasalahan-permasalahan tersebut tidak hanya meningkatkan beban teknis selama praktikum, tetapi juga mengurangi efektivitas proses pembelajaran karena sebagian waktu perkuliahan digunakan untuk menyelesaikan kendala konfigurasi dibandingkan mempelajari materi praktikum.

Sejumlah penelitian sebelumnya telah mengkaji pengembangan infrastruktur server dan jaringan untuk mendukung sistem pembelajaran berbasis komputasi. Jupyter Notebook dan JupyterHub menjadi solusi yang banyak diadopsi karena kemampuannya menyediakan lingkungan pemrograman berbasis web yang terintegrasi [4]. Studi oleh berbagai peneliti menunjukkan bahwa implementasi JupyterHub pada lingkungan *cloud* mampu memberikan skalabilitas tinggi dan kemudahan manajemen pengguna melalui pendekatan containerisasi dan orkestrasi seperti Kubernetes [5], [6], [7]. Selain itu, distribusi seperti The Littlest JupyterHub dirancang untuk skenario sumber daya terbatas dengan pendekatan instalasi yang lebih ringan [8], [9]. Penelitian lain menekankan pentingnya konfigurasi *resource management*, autentikasi pengguna (misalnya PAM) [10], serta mekanisme *idle-culling* untuk menjaga stabilitas sistem saat digunakan secara simultan [11]. Namun demikian, sebagian besar studi tersebut berfokus pada institusi dengan infrastruktur besar atau berbasis *cloud*, sehingga belum sepenuhnya merepresentasikan kondisi perguruan tinggi skala menengah dengan keterbatasan sumber daya lokal. Gap yang masih tersisa terletak pada kurangnya kajian implementasi JupyterHub pada server lokal (*on-premise*) dengan pendekatan optimasi yang adaptif, serta minimnya evaluasi kuantitatif performa multiuser dalam konteks pembelajaran praktikum di lingkungan tersebut.

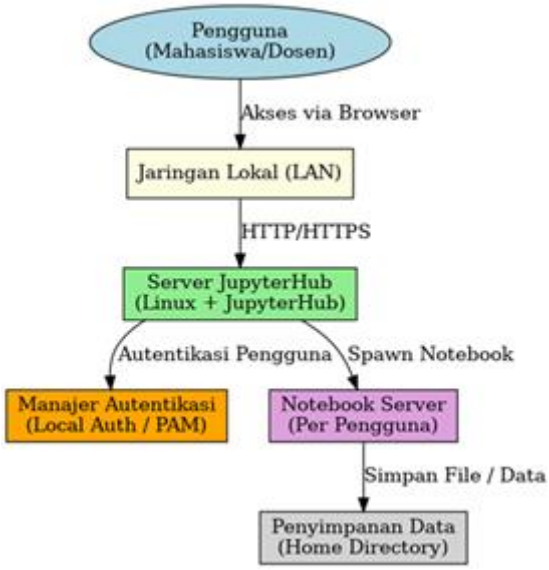
Berbeda dengan penelitian sebelumnya yang berfokus pada implementasi JupyterHub pada lingkungan *cloud* atau infrastruktur komputasi berskala besar, penelitian ini mengusulkan model implementasi JupyterHub yang terintegrasi dengan *cluster* komputasi berbasis Dask, otomatisasi manajemen pengguna, serta optimasi sumber daya menggunakan *Idle culler* pada server lokal dengan kapasitas terbatas. Selain itu, penelitian mengenai implementasi JupyterHub yang terintegrasi dengan komputasi terdistribusi berbasis Dask pada server lokal perguruan tinggi masih sangat terbatas, khususnya yang mengevaluasi distribusi komputasi, pemanfaatan sumber daya server, serta ketahanan *cluster* pada lingkungan laboratorium pendidikan.

Oleh karena itu, penelitian ini tidak hanya berfokus pada implementasi JupyterHub, tetapi juga mengevaluasi kemampuan integrasinya dengan *cluster* Dask dalam mendukung komputasi terdistribusi serta efisiensi pengelolaan sumber daya server pada lingkungan laboratorium pendidikan.

2. Metodologi

Penelitian ini menggunakan pendekatan eksperimental untuk mengimplementasikan dan mengevaluasi layanan JupyterHub sebagai platform praktikum Python berbasis server lokal di Laboratorium Informatika Universitas Janabadra. Metodologi penelitian meliputi analisis kebutuhan sistem, perancangan arsitektur layanan, implementasi komponen pendukung, serta validasi dan evaluasi sistem berdasarkan kondisi operasional server. Evaluasi dilakukan melalui pengamatan terhadap status layanan, mekanisme autentikasi, pengelolaan pengguna, implementasi *cluster* komputasi, serta pemanfaatan sumber daya server selama sistem dijalankan.

Analisis kebutuhan dilakukan untuk mengidentifikasi kebutuhan fungsional dan nonfungsional sistem praktikum Python berbasis server. Dari aspek fungsional, sistem harus mampu menyediakan layanan Jupyter Notebook berbasis web, mendukung autentikasi pengguna menggunakan Pluggable Authentication Module (PAM), membentuk notebook server yang terisolasi (isolated notebook server) untuk setiap pengguna, serta menyediakan lingkungan Python yang seragam dengan penyimpanan data yang bersifat persisten pada direktori *home* masing-masing pengguna. Dari aspek nonfungsional, sistem dirancang agar memiliki tingkat ketersediaan layanan yang tinggi, mudah dikelola oleh administrator, serta mampu memanfaatkan sumber daya server secara efisien selama pelaksanaan praktikum.



Gambar 1. Arsitektur Sistem JupyterHub Berbasis Server Lokal

Gambar 1 menunjukkan arsitektur sistem praktikum Python berbasis JupyterHub yang diimplementasikan pada server Laboratorium Informatika Universitas Janabadra. Seluruh pengguna mengakses layanan menggunakan peramban (*web browser*) melalui jaringan lokal (LAN) menggunakan protokol HTTP/HTTPS. Seluruh proses komputasi dilakukan secara terpusat pada server sehingga pengguna tidak perlu melakukan instalasi Python maupun pustaka pendukung pada komputer masing-masing.

Selain menyediakan layanan notebook berbasis web, sistem juga mengintegrasikan *cluster* komputasi terdistribusi berbasis Dask untuk mendukung eksekusi komputasi paralel [11], [12]. *Cluster* terdiri atas 1 head *node* dan 7 *worker node*, di mana head *node* juga berfungsi sebagai *worker* sehingga total terdapat 8 *worker* aktif. Layanan JupyterHub dan Dask *Scheduler* dijalankan pada head *node*, sedangkan layanan Dask *Worker* dijalankan pada seluruh *node cluster* [13]. Notebook yang dijalankan pengguna berperan sebagai Dask Client yang mengirimkan tugas komputasi kepada *Scheduler* untuk kemudian didistribusikan secara otomatis ke *worker* yang tersedia. Arsitektur ini memungkinkan proses komputasi dilakukan secara paralel sehingga beban pemrosesan tidak hanya terpusat pada satu server, tetapi dibagi ke beberapa *node* sesuai dengan ketersediaan sumber daya komputasi.

Untuk menjaga ketersediaan sumber daya server selama praktikum, sistem dikonfigurasi menggunakan mekanisme *Idle culler* [14], [15]. Layanan ini memantau aktivitas notebook setiap pengguna dan secara otomatis menghentikan notebook yang tidak aktif setelah melewati batas waktu tertentu (idle timeout). Pada penelitian ini, *Idle culler* dikonfigurasi dengan waktu timeout selama 1200 detik dan melakukan pemeriksaan aktivitas setiap 60 detik. Penerapan *Idle culler* bertujuan mencegah notebook yang tidak lagi digunakan tetap mengonsumsi memori dan sumber daya prosesor.

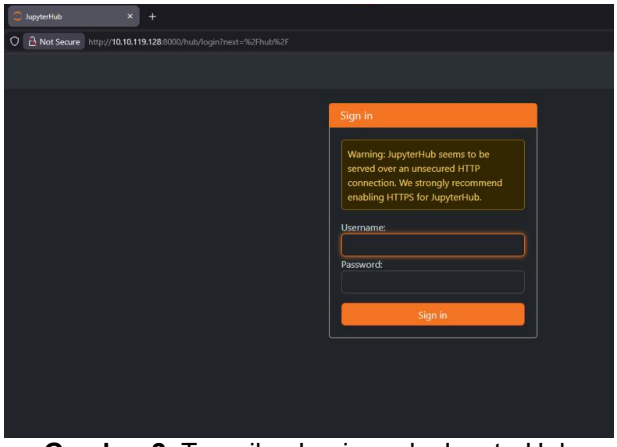
Setelah implementasi selesai, dilakukan validasi terhadap seluruh komponen sistem meliputi layanan JupyterHub, autentikasi PAM, Dask *Scheduler*, Dask *Worker*, serta mekanisme otomatisasi pengguna [10]. Validasi dilakukan menggunakan utilitas systemctl, log layanan JupyterHub, *Dashboard* Dask, serta pengujian akses melalui notebook. Evaluasi sistem dilakukan melalui observasi terhadap kondisi operasional *cluster*. Parameter yang diamati

meliputi status layanan, jumlah *worker* aktif, distribusi task, penggunaan CPU, penggunaan memori, bandwidth, jumlah file descriptor, serta kemampuan *worker* bergabung kembali ke *cluster* setelah proses komputasi selesai.

3. Hasil dan Pembahasan

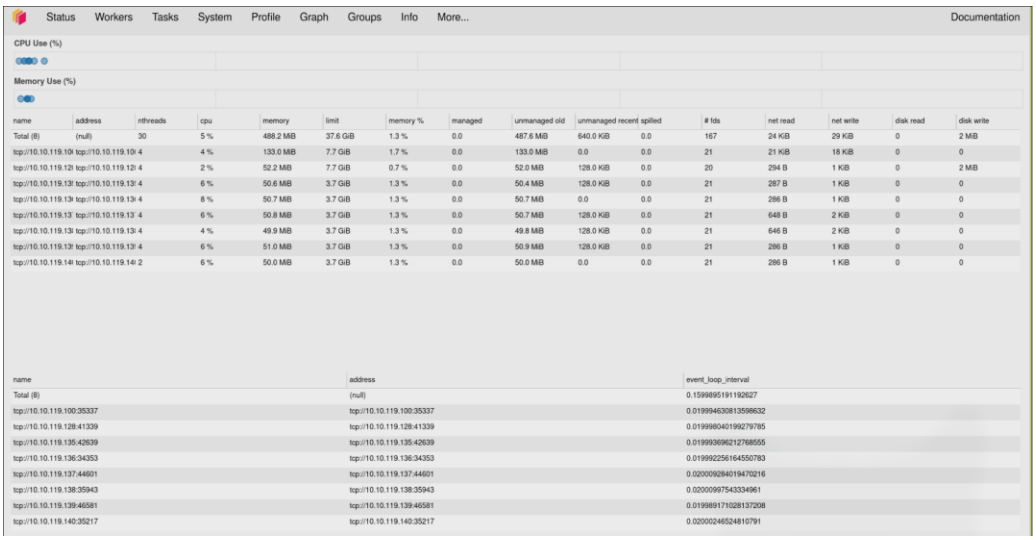
3.1. Hasil Implementasi Sistem

Implementasi JupyterHub pada server laboratorium berhasil dilakukan sesuai dengan rancangan arsitektur sistem yang telah ditetapkan. Sistem dapat diakses melalui jaringan lokal menggunakan browser tanpa memerlukan instalasi tambahan pada sisi pengguna. Integrasi dengan Dask juga berhasil dilakukan untuk mendukung komputasi paralel, yang memungkinkan distribusi beban kerja pada beberapa *worker*. Seluruh komponen utama sistem, termasuk proxy, hub, dan notebook server, berjalan secara stabil. Mekanisme autentikasi berbasis PAM mampu membatasi akses hanya pada pengguna yang terdaftar, sedangkan sistem penyimpanan berbasis direktori *home* memungkinkan pengguna menyimpan hasil pekerjaan secara persisten. Hasil implementasi sistem ditunjukkan melalui tampilan antarmuka JupyterHub pada Gambar 2.



Gambar 2. Tampilan Login pada JupyterHub

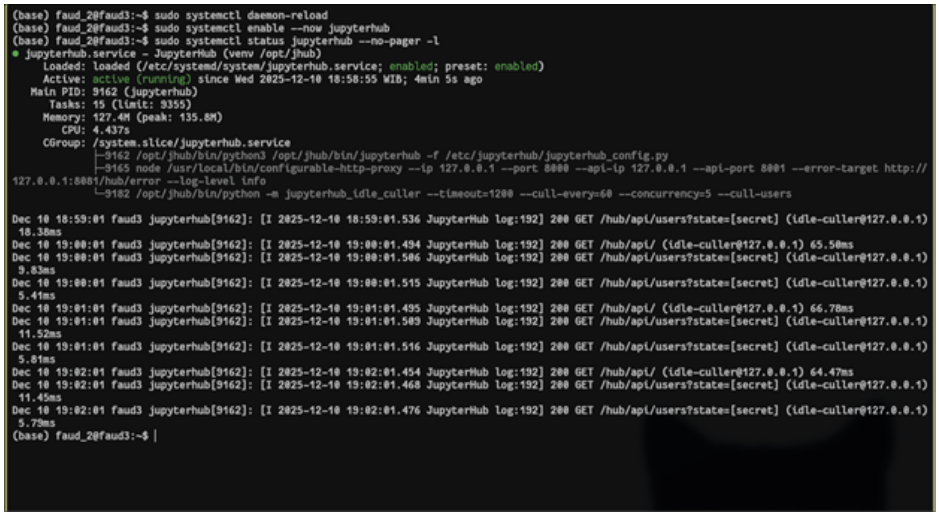
Untuk mendukung komputasi paralel pada praktikum Python, sistem diintegrasikan dengan *cluster* Dask yang terdiri atas satu head *node* dan tujuh *worker node*. Head *node* menjalankan layanan JupyterHub sekaligus Dask *Scheduler*, sedangkan seluruh *worker* bertugas mengeksekusi komputasi yang didistribusikan oleh *scheduler*. *Dashboard* Dask pada Gambar 3 menunjukkan bahwa seluruh *worker* berhasil melakukan registrasi ke *scheduler* sehingga *cluster* siap digunakan.



Gambar 3. *Dashboard* Dask (menu *Worker*s) yang menunjukkan registrasi *worker*.

3.2. Validasi Implementasi Sistem.

Hasil validasi menunjukkan bahwa layanan *Idle culler* berhasil dijalankan sebagai *service* dan berada pada status aktif bersama layanan JupyterHub sebagaimana terlihat pada Gambar 4. Log tersebut bahwa layanan JupyterHub berada pada status *active* (*running*) dan *enabled*. Selain itu, proses utama JupyterHub, *Configurable* HTTP Proxy, serta layanan *Idle culler* berhasil dijalankan secara bersamaan sebagai *service* yang dikelola oleh systemd. Log layanan menunjukkan respon HTTP 200, yang mengindikasikan bahwa seluruh layanan berhasil berjalan sesuai dengan konfigurasi sistem.

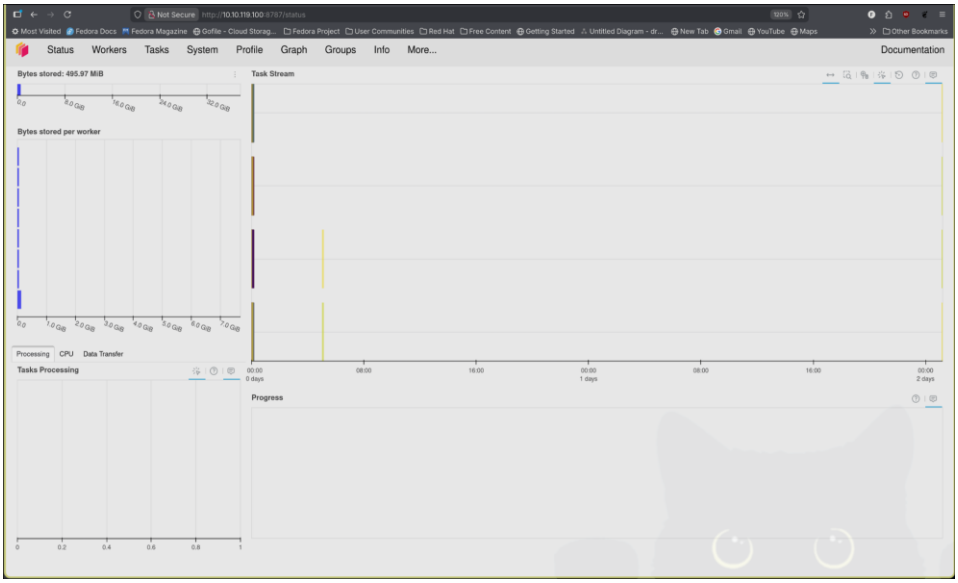


Gambar 4. Status layanan JupyterHub menggunakan utilitas systemctl

3.4 Hasil Pengujian Eksperimen

a. Pengujian Distribusi Komputasi

Pengujian distribusi komputasi dilakukan untuk memverifikasi bahwa arsitektur *cluster* yang telah diimplementasikan mampu mendistribusikan tugas komputasi dari notebook pengguna ke seluruh Dask *Worker* melalui Dask *Scheduler*. Pada pengujian ini, notebook yang dijalankan melalui JupyterHub dikonfigurasikan sebagai Dask Client yang mengirimkan operasi komputasi berbasis Dask Array ke *scheduler*. Selanjutnya *scheduler* mendistribusikan task tersebut ke *worker* yang tersedia sesuai dengan kondisi sumber daya masing-masing *node*.



Gambar 5. Visualisasi distribusi task komputasi pada *Dashboard* Dask.

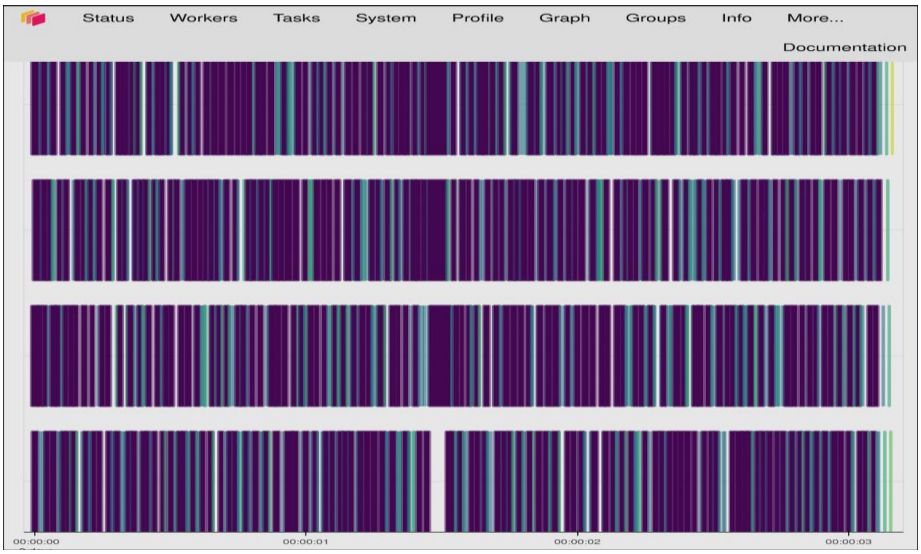
Berdasarkan hasil pengamatan pada Gambar 5, setiap task yang dikirimkan dari notebook berhasil diterima oleh *scheduler* dan didistribusikan ke *worker* dalam *cluster*. *Dashboard* Dask pada menu Tasks memperlihatkan bahwa task diproses secara paralel dengan waktu eksekusi yang relatif singkat, tanpa ditemukan task yang gagal (failed task) maupun timeout. Hal ini menunjukkan bahwa mekanisme penjadwalan (task scheduling) berjalan sesuai dengan rancangan sistem.

Keberhasilan distribusi task menunjukkan bahwa notebook tidak lagi menjalankan seluruh proses komputasi pada head *node*, melainkan hanya berfungsi sebagai antarmuka (client) yang mengirimkan permintaan komputasi. Selanjutnya, *scheduler* bertanggung jawab menentukan *worker* yang akan mengeksekusi setiap task sehingga beban komputasi dapat didistribusikan secara lebih merata pada seluruh *node* yang tersedia. Pendekatan ini meningkatkan efisiensi pemanfaatan sumber daya server dibandingkan model komputasi yang seluruh prosesnya dijalankan pada satu mesin.

Hasil pengujian ini mengonfirmasi bahwa integrasi antara JupyterHub dan Dask telah berhasil diimplementasikan sesuai dengan rancangan arsitektur yang diusulkan. Selain membuktikan keberhasilan komunikasi antara notebook, *scheduler*, dan *worker*, pengujian ini juga menjadi dasar bagi evaluasi pada tahap berikutnya, yaitu pengamatan terhadap pemanfaatan sumber daya server selama proses komputasi berlangsung.

b. Pengujian Eksekusi oleh Worker

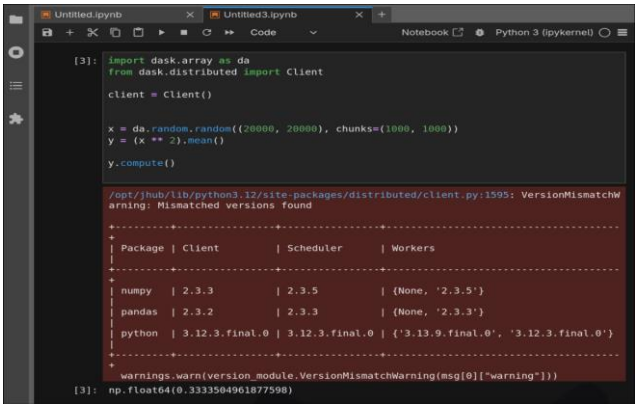
Setelah task berhasil didistribusikan, dilakukan pengamatan terhadap aktivitas *worker* selama proses komputasi berlangsung. Gambar menunjukkan bahwa *worker* secara aktif memproses task yang diterima dari *scheduler* hingga proses komputasi selesai. Hasil pada Gambar 6 ini membuktikan bahwa proses eksekusi benar-benar dilakukan oleh *worker* dalam *cluster* dan bukan hanya dijalankan pada head *node*.



Gambar 6. Aktivitas *worker* yang didistribusikan oleh Dask *Scheduler*.

c. Pengujian Pengembalian Hasil Komputasi

Tahap berikutnya adalah memastikan bahwa hasil komputasi yang telah diproses oleh *worker* dapat dikembalikan kepada notebook pengguna. Gambar 7 menunjukkan bahwa output komputasi berhasil ditampilkan pada notebook tanpa kesalahan, dengan nilai keluaran sesuai hasil perhitungan Dask. Hal ini membuktikan bahwa komunikasi antara notebook, *scheduler*, dan *worker* berlangsung secara normal hingga proses komputasi selesai.

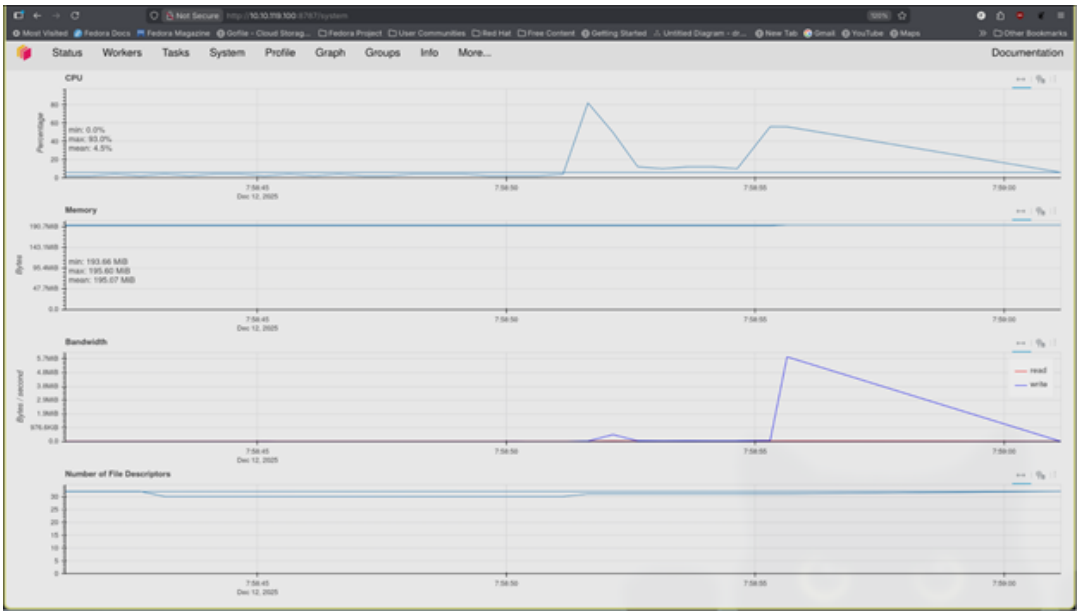


Gambar 7. Output komputasi Dask pada notebook setelah proses eksekusi selesai.

d. Pengujian Pemanfaatan Sumber Daya Server

Setelah proses distribusi dan eksekusi komputasi berhasil dilakukan, dilakukan pemantauan terhadap kondisi operasional *cluster* menggunakan *Dashboard* Dask pada menu System. Pengamatan bertujuan mengevaluasi dampak aktivitas komputasi terhadap penggunaan sumber daya server.

Berdasarkan Gambar 8, penggunaan CPU mengalami peningkatan ketika *scheduler* mendistribusikan task kepada *worker*, kemudian kembali menurun setelah proses komputasi selesai. Hal serupa juga terjadi pada aktivitas bandwidth jaringan yang meningkat selama komunikasi antara *scheduler* dan *worker* berlangsung. Sebaliknya, penggunaan memori relatif stabil sepanjang proses pengujian sehingga tidak ditemukan indikasi kebocoran memori (memory leak). Jumlah file descriptor juga berada pada kondisi yang stabil, yang menunjukkan bahwa koneksi antar komponen sistem dapat dipertahankan dengan baik selama layanan berjalan.

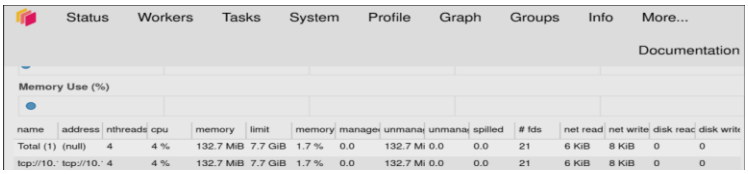


Gambar 8. Monitoring penggunaan CPU, memori, bandwidth, dan file descriptor menggunakan *Dashboard* Dask (menu System).

e. Pengujian Ketahanan terhadap Gangguan

Pengujian dilakukan dengan memberikan beban komputasi yang lebih tinggi untuk mengamati kemampuan *cluster* dalam mempertahankan layanan ketika sumber daya mulai terbatas. Hasil pada Gambar 9 menunjukkan bahwa Dask *Scheduler* tetap berada pada kondisi aktif meskipun beberapa *worker* mengalami penghentian proses (killed) akibat keterbatasan perangkat keras. Kondisi ini menunjukkan bahwa *scheduler* tetap mampu mempertahankan layanan sehingga proses komputasi tidak berhenti secara keseluruhan. Namun demikian,

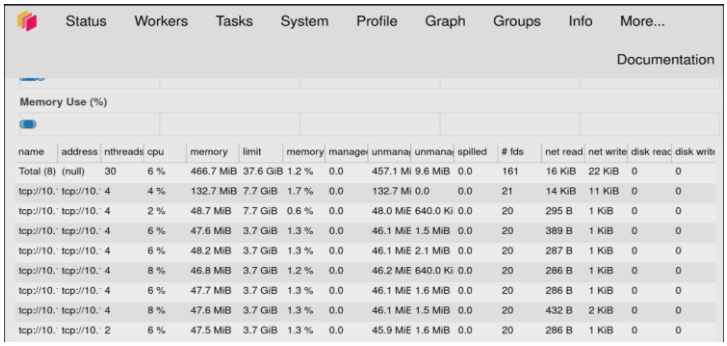
keterbatasan kapasitas memori dan prosesor masih menjadi faktor pembatas ketika beban komputasi meningkat secara signifikan.



Gambar 9. Kondisi *cluster* saat pengujian ketahanan sistem.

e. Pengujian Pemulihan Worker

Pengujian terakhir dilakukan untuk mengevaluasi kemampuan *worker* bergabung kembali ke dalam *cluster* setelah proses komputasi selesai. Hasil pengamatan menunjukkan bahwa *worker* yang sebelumnya keluar dari *cluster* berhasil melakukan registrasi ulang dan kembali muncul pada *dashboard* monitoring tanpa memerlukan konfigurasi tambahan. Hasil pada Gambar 10 menunjukkan bahwa mekanisme pemulihan (recovery) pada *cluster* berjalan dengan baik sehingga layanan dapat kembali ke kondisi normal setelah terjadi gangguan sementara.



Gambar 10. *Worker* berhasil melakukan registrasi ulang dan bergabung kembali ke dalam *cluster* setelah proses komputasi selesai.

3.3 Pembahasan

Hasil implementasi menunjukkan bahwa konsep lingkungan praktikum Python berbasis JupyterHub mampu mengatasi permasalahan yang diidentifikasi pada pendahuluan, yaitu perbedaan konfigurasi perangkat, ketidaksesuaian versi pustaka Python, serta kesulitan pengelolaan lingkungan praktikum secara terpusat. Seluruh proses komputasi dipindahkan ke server sehingga mahasiswa hanya memerlukan peramban web untuk mengakses notebook dengan lingkungan Python yang seragam. Pendekatan ini menghilangkan ketergantungan terhadap konfigurasi perangkat pengguna sekaligus meningkatkan konsistensi pelaksanaan praktikum. Temuan ini memperkuat penelitian Flemming [4] yang menunjukkan bahwa JupyterHub efektif digunakan sebagai platform praktikum berbasis server, serta sejalan dengan penelitian Bascuñana dkk. [6] yang menyatakan bahwa notebook interaktif mampu meningkatkan efektivitas pembelajaran melalui lingkungan kerja yang lebih konsisten dan mudah direproduksi.

Kontribusi utama penelitian ini terletak pada integrasi cluster komputasi berbasis Dask ke dalam layanan JupyterHub pada server lokal. Hasil pengujian menunjukkan bahwa notebook berhasil berfungsi sebagai Dask Client, sedangkan *scheduler* mampu mendistribusikan task secara otomatis kepada seluruh *worker* hingga hasil komputasi dikembalikan ke notebook tanpa kesalahan. Keberhasilan registrasi *worker*, distribusi task, eksekusi komputasi, dan *worker* recovery menunjukkan bahwa arsitektur yang diusulkan tidak hanya mendukung layanan notebook multiuser, tetapi juga menyediakan komputasi terdistribusi yang dapat dimanfaatkan untuk praktikum data mining, machine learning, maupun data science. Hasil ini memperluas implementasi JupyterHub yang sebelumnya lebih banyak diterapkan pada lingkungan *cloud* dan high-performance computing [8], [10], menjadi solusi yang layak diterapkan pada laboratorium pendidikan dengan sumber daya yang terbatas.

Monitoring menggunakan Dashboard Dask memperlihatkan bahwa penggunaan CPU, memori, bandwidth, dan file descriptor berada pada kondisi yang stabil selama proses komputasi. Selain itu, penerapan *Idle culler* berhasil menghentikan notebook yang tidak aktif sehingga

sumber daya server dapat digunakan kembali oleh pengguna lain tanpa intervensi administrator. Temuan ini menunjukkan bahwa efisiensi sistem tidak hanya dipengaruhi oleh spesifikasi perangkat keras, tetapi juga oleh konfigurasi layanan yang mampu mengoptimalkan alokasi sumber daya. Hasil tersebut mendukung penelitian Reppin dkk. [11] dan Stetzler dkk. [14] yang menekankan pentingnya pengelolaan sumber daya dalam menjaga stabilitas layanan notebook pada lingkungan multiuser.

Pengujian *fault tolerance* menunjukkan bahwa Dask *Scheduler* tetap aktif ketika sebagian *worker* mengalami penghentian proses, sedangkan *worker* dapat bergabung kembali ke dalam cluster setelah kondisi sistem normal. Walaupun penelitian ini masih terbatas pada lingkungan laboratorium dengan jumlah *node* yang relatif kecil, hasil tersebut menunjukkan bahwa arsitektur yang diusulkan telah memiliki kemampuan *fault tolerance* dasar yang memadai untuk mendukung praktikum reguler. Dengan demikian, penelitian ini tidak hanya mengonfirmasi efektivitas implementasi JupyterHub pada server lokal, tetapi juga memberikan kontribusi berupa integrasi Dask, otomatisasi manajemen pengguna, optimasi sumber daya menggunakan *Idle culler*, serta evaluasi ketahanan layanan pada laboratorium pendidikan. Model implementasi ini berpotensi direplikasi pada perguruan tinggi lain yang memiliki keterbatasan infrastruktur dan dapat dikembangkan lebih lanjut melalui integrasi GPU, Docker, atau Kubernetes untuk meningkatkan skalabilitas layanan.

4. Simpulan

Penelitian ini berhasil mengimplementasikan JupyterHub sebagai platform praktikum Python berbasis server lokal di Laboratorium Informatika Universitas Janabadra melalui arsitektur client-server yang terintegrasi dengan autentikasi PAM, manajemen pengguna otomatis, serta mekanisme *Idle culler* untuk mengoptimalkan pemanfaatan sumber daya server. Hasil validasi menunjukkan bahwa seluruh komponen sistem, meliputi layanan JupyterHub, autentikasi pengguna, notebook server, dan monitoring *cluster* menggunakan *Dashboard* Dask, berfungsi sesuai dengan rancangan serta mampu menyediakan lingkungan praktikum yang seragam, mudah dikelola, dan stabil. Kontribusi penelitian ini terletak pada penyediaan model implementasi JupyterHub yang dioptimalkan untuk server lokal perguruan tinggi dengan sumber daya terbatas, sehingga dapat menjadi referensi bagi institusi lain dalam mengembangkan laboratorium virtual berbasis server. Penelitian selanjutnya disarankan untuk menguji skalabilitas sistem pada jumlah pengguna yang lebih besar serta mengintegrasikan teknologi orkestrasi kontainer dan GPU guna mendukung komputasi yang lebih kompleks.

Daftar Referensi

[1] M. H. Maulana, “Python Bahasa Pemograman Yang Ramah Bagi Pemula,” *JISCO : Journal of Information System and Computing*, vol. 2, no. 2, pp. 73–78, Dec. 2024, doi: 10.30631/jisco.v2i2.105.

[2] “The Littlest JupyterHub,” The Littlest JupyterHub. Accessed: Aug. 04, 2026. [Online]. Available: https://tljh.jupyter.org/en/latest/index.html

[3] N. S. N. Az-zahrani, H. K. A. Eloi, F. Salim, A.-Z. A. Ramadhani, C. Meysyanti, and L. N. A. Purwantiningsih, *Python untuk Analisis Data*. SIEGA Publisher, 2025.

[4] J. Flemming, “JupyterHub and autograding on bare-metal lab servers,” Westsächsische Hochschule Zwickau, 2022. doi: 10.34806/6F2W-V119.

[5] T. P. Rezeki and M. I. P. Nasution, “Integrasi Data Base Berbasis Cloud Untuk Skalabilitas Bisnis: Studi Kasus Netflix,” *Journal Sains Student Research*, vol. 3, no. 3, pp. 328–340, May 2025, doi: 10.61722/jssr.v3i3.4743.

[6] J. Bascuñana, S. León, M. González-Miquel, E. J. González, and J. Ramírez, “Impact of Jupyter Notebook as a tool to enhance the learning process in chemical engineering modules,” *Education for Chemical Engineers*, vol. 44, pp. 155–163, Jul. 2023, doi: 10.1016/j.ece.2023.06.001.

[7] M. Rocklin, “Dask: Parallel Computation with Blocked algorithms and Task Scheduling,” presented at the Python in Science Conference, Austin, Texas, 2015, pp. 126–132. doi: 10.25080/Majora-7b98e3ed-013.

[8] J. Stubbs *et al.*, “Integrating Jupyter into Research Computing Ecosystems: Challenges and Successes in Architecting JupyterHub for Collaborative Research Computing Ecosystems,” in *Practice and Experience in Advanced Research Computing 2020: Catch the Wave*, in

PEARC '20. New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 91–98. doi: 10.1145/3311790.3396648.

[9] P. Prathanrat and C. Polprasert, "Performance Prediction of Jupyter Notebook in JupyterHub using Machine Learning," *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, pp. 157–162, Oct. 2018, doi: 10.1109/ICIIBMS.2018.8550030.

[10] D. Li, R. Pyke, and R. Jiang, "A Scalable Cloud-based Architecture to Deploy JupyterHub for Computational Social Science Research," in *Practice and Experience in Advanced Research Computing*, Boston MA USA: ACM, Jul. 2021, pp. 1–4. doi: 10.1145/3437359.3465591.

[11] J. Reppin *et al.*, "Interactive analysis notebooks on DESY batch resources: Bringing Juypter to HTCondor and Maxwell at DESY," *Comput Softw Big Sci*, vol. 5, no. 1, p. 16, Dec. 2021, doi: 10.1007/s41781-021-00058-y.

[12] S. Ibrahim, T. Stitt, M. Larsen, and C. Harrison, "Interactive in situ visualization and analysis using Ascent and Jupyter," in *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, in ISAV '19. New York, NY, USA: Association for Computing Machinery, Jan. 2020, pp. 44–48. doi: 10.1145/3364228.3364232.

[13] S. Böhm and J. Beránek, "Runtime vs Scheduler: Analyzing Dask's Overheads," in *2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, Nov. 2020, pp. 1–8. doi: 10.1109/WORKS51914.2020.00006.

[14] Steven Stetzler, Mario Jurić, Kyle Boone, Andrew Connolly, Colin T. Slater, and Petar Zečević, "The Astronomy Commons Platform: A Deployable Cloud-based Analysis Platform for Astronomy," *The Astronomical Journal*, vol. 164, no. 68, p. (18 pp), 2022, doi: https://doi.org/10.3847/1538-3881/ac77fb.

[15] J. Rolf, M. Wolf, and D. Gerhard, "Concept to Manage and Grade Python Programming Assignments in Large Cohorts," in *Open Science in Engineering*, M. E. Auer, R. Langmann, and T. Tsiatsos, Eds., Cham: Springer Nature Switzerland, 2023, pp. 737–747. doi: 10.1007/978-3-031-42467-0_69.