2026 IEEE/SICE International Symposium on System Integration (SII)
January 11-14, 2026. Cancún, México

Fog Computing-Enabled Multi-User Interaction System using Soft Robotic Gloves

Kadir Ozlem[1], Asli Tuncay Atalay[2], Ozgur Atalay[2] and Gökhan Ince[1,*]

***Abstract*— Soft robotics, with its flexible and lightweight structures, offers innovative solutions in fields such as medicine, rehabilitation, and assistive technologies. The development of textile-based soft robotics has significantly reduced integration challenges in wearable technologies. This study presents a novel fog computing architecture designed to support multi-user and multi-device operations used in textile-based soft robotics systems. For patients experiencing muscle weakness in one hand, Textile-based Internet of Things (T-IoT) gloves enable simultaneous bilateral actions such as grasping, lifting, and carrying through a mirroring technique. Sensor data from a sensing T-IoT glove worn on the healthy hand are processed by the fog computing system and transmitted to an actuating T-IoT glove worn on the impaired hand, enabling coordinated movement of both hands. By integrating various concurrency and interprocess communication techniques into worker devices, the system can serve up to 10 users with a single worker and up to 26 users with three workers operating simultaneously. In this way, the proposed system provides real-time computational services for multi-user scenarios while also delivering an assistive technology for patients.**

I. INTRODUCTION

Soft robotics refers to flexible and compliant robotic systems that mimic biological structures [1]. Developed as an alternative to rigid systems, this technology finds applications in diverse fields such as medicine, rehabilitation, industrial automation, wearable technologies, search and rescue, agriculture, underwater operations, and space exploration [2]. Soft robotic actuators can be fabricated from a wide variety of materials, including silicone, elastomers, hydrogels, shape memory alloys [3]. Textile materials, which are lightweight, skin-compatible, flexible, and comfortable, feature desired characteristics in daily life and have also become prominent in actuator development [4]. Particularly through knitting technology, actuators can be produced rapidly, repeatably, and with minimal human error [5].

This research is partially supported by the European Research Council (ERC) under the European Union's Horizon Research and Innovation Programme (Grant No: 101042402) and the Scientific Research Project Unit of Istanbul Technical University(Grant No: MDK-2021-43137). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

[1]Kadir Ozlem and Gökhan Ince (*corresponding author) are with the Faculty of Computer and Informatics Engineering, Computer Engineering Department, Istanbul Technical University, 34469 Istanbul, Turkey. kadir.ozlem@itu.edu.tr; gokhan.ince@itu.edu.tr

[2]Asli Tuncay Atalay and Ozgur Atalay are with the Faculty of Textile Technologies and Design, Textile Engineering Department, Istanbul Technical University, 34437 Istanbul, Turkey. aatalay@itu.edu.tr; atalayoz@itu.edu.tr

Although soft robotics has gained traction across many domains, most existing research has focused on materials [6] and control systems [7]. With increasing applications, the demand for computational systems in this field has grown, but remains underexplored. Due to its characteristics such as low latency, reduced bandwidth usage, real-time capability, mobility support, and energy efficiency [8], fog computing offers a promising solution to address the computational requirements of soft robotic systems. Soft robotic systems are increasingly implemented as wearable devices for rehabilitation and daily assistance, where embedded hardware is limited in both power and computational capacity. These constraints make low-latency offloading essential for running control and machine learning tasks, thereby justifying the need for a distributed fog infrastructure.

Studies on fog computing have been conducted in various domains such as smart cities [9], connected vehicles [10], smart homes [11], healthcare [12], and electronic textiles [13], yet no work has been reported in the context of soft robotics. Regarding concurrency techniques, existing research has explored single-thread [14], multi-thread [14], and multi-process [15] approaches. In terms of Inter-Process Communication (IPC), methods such as shared memory [16], file-based communication [17], and Redis (socket-based) [18] have been employed, as well as fog computing frameworks that integrate these mechanisms. However, to the best of our knowledge, no study has investigated a fog computing system incorporating different combinations of concurrency and IPC techniques with a focus on the number of users it can serve simultaneously. Most existing works primarily evaluate the overall system response time, without addressing how these techniques differ in their impact on multi-user performance.

The primary objective of this study is to develop a fog computing architecture capable of serving multiple users simultaneously by implementing different concurrency methods within the fog device. In the proposed case study, signals collected from a glove embedded with e-textile sensors are processed into control commands for textile-based soft robotic actuators, and the system performance is evaluated. This approach simplifies the hardware requirements on both the sensing and actuating ends while ensuring efficient utilization of computational resources.

In rehabilitation settings, a single fog node often needs to handle several patients at once, and increasing user count directly impacts latency. Therefore, understanding multi-user scalability through real-device evaluations is essential, as simulations alone cannot reveal performance limits.

iFogSim has been widely used to study resource management and QoS optimization, and while its scenarios are often simplified, its modular design allows configuring multiple sensors, actuators, and applications [19]. YAFS extends these capabilities by supporting dynamic topology modeling with multiple devices and users, enabling evaluation of latency, response time, and energy consumption under more realistic conditions [20]. More recently, LEAF provides large-scale energy-aware simulations that can handle thousands of devices and tasks, allowing systematic exploration of multi-user workloads at scale [21]. In this study, a traffic generator was developed to enable real-time testing under multi-user conditions. We expect a multiprocessing approach with FIFO IPC to deliver the best performance under multi-user conditions, providing a complementary perspective to simulation-based studies through validation on real fog devices. While the proposed architecture is tailored to soft robotic wearables, the primary contribution of this work is the evaluation of concurrency and IPC methods for achieving scalable multi-user performance in fog environments. The application scenario provides a representative case demonstrating the practical relevance of the proposed framework.

II. Multi-User Communication System for Soft Robotics

This study proposes a fog computing architecture designed for assistive soft robotic systems, as illustrated in Figure 1. Signals acquired from a sensing Textile-based Internet of Things (T-IoT) glove are processed using machine learning techniques within the fog computing environment and converted into control commands for a corresponding actuating T-IoT glove worn on a hand with muscle weakness, enabling movement generation. This setup allows individuals with unilateral muscle impairment to perform daily actions such as grasping, lifting, and transporting heavy or bulky objects through mirroring techniques using both hands in synchronization. The system integrates sensing and actuating T-IoT gloves within a coordinated fog computing infrastructure, where worker devices employ various concurrency and IPC techniques to support multi-user operation and maximize the number of users the system can serve.

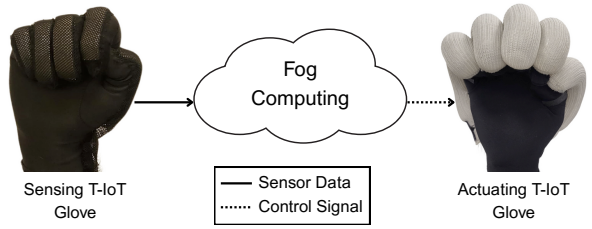

Fig. 1. System Overview.

A. T-IoT Gloves

In this study, sensing [22] and actuating [23] T-IoT gloves were employed to enable patients with muscle weakness in one hand to perform simultaneous movements. The sensing T-IoT glove is equipped with textile-based capacitive stretch sensors that detect finger movements via a microcontroller and transmit the data through Bluetooth.

The actuating T-IoT glove incorporates textile-based actuators developed using a 3D knitting technique. Each actuator, featuring two air chambers, is capable of performing both extension and flexion movements. A pneumatic control system allows independent control of each finger.

B. Fog Computing Framework

Sensor data from the sensing T-IoT glove is processed by worker devices in the fog computing infrastructure to generate control signals. The architecture was built using the FogETex framework [13]. To convert sensor data into control signals, the decision tree model, identified in a previous study [22] as the most effective in both accuracy and response time, was employed. By integrating the sensing and actuating T-IoT gloves through the fog infrastructure, this study aims to reduce response time and increase the number of users that can be served simultaneously through different concurrency methods.

Due to the wide range of existing machine learning libraries and the relatively simpler development process, the computing module of the FogETex framework was developed using Python to enable other developers to easily write applications for the framework [13]. The biggest obstacle to effectively utilizing processor cores is the Global Interpreter Lock (GIL) in the Python interpreter. The GIL mechanism only allows one thread to execute Python code at a time, thus preventing concurrency issues that may arise in memory and data management. However, it also introduces performance issues in parallel operations and CPU-intensive tasks [24]. Therefore, performance improvements using different concurrency techniques are required in this context.

C. Concurrency Control Techniques

The FogETex framework is inherently designed to serve multiple users efficiently. Effective utilization of processor cores is crucial for supporting more users. In this study, we incorporate single-threaded, multi-threaded, and multi-process concurrency methods to the FogETex framework. All three methods utilize the WebSocket IPC mechanism.

Since the socket server and the computing module operate in separate processes on computing service devices, IPC is critical for system performance. In addition to the WebSocket IPC method, the FogETex framework is also extended to support First In First Out IPC (FIFO IPC) and RESTful API IPC methods through the multi-process concurrency approach. In total, five different concurrency and IPC methods can be configured in the proposed system. In all methods, users send computation requests to the computing service devices through a WebSocket connection.

1) Single-threaded data processing via WebSocket IPC: Figure 2 illustrates the data communication flow of this method. Its structure is simpler than in other methods.

Data received by the socket server is forwarded to the computing module via WebSocket. The socket client module within the computing module adds the data to a queue. Once the computation requests that arrived earlier are completed, the data is processed using the developed model and returned

to the user through the socket client and socket server. Since all operations within the computing module occur in the same process, data is transferred directly through memory.

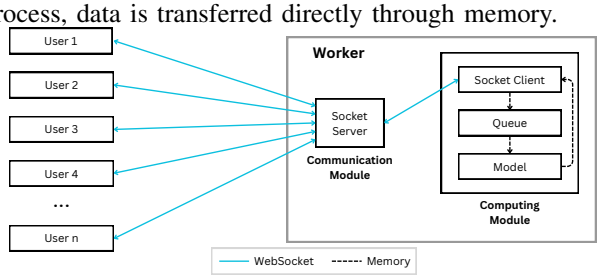

Fig. 2. Single-threaded Data Processing via WebSocket IPC.

2) Multi-threaded data processing via WebSocket IPC: The second method is multi-threaded data processing via the WebSocket IPC method (Figure 3). In the single-threaded method, each incoming computation request is added to the queue, and the new request must wait for the completion of the earlier ones. In the multi-threaded method, however, computation requests in separate threads do not have to wait for each other. This aims to reduce queue times.

In this method, each thread has a dedicated socket client, queue, and model. This allows computation requests in different threads to be executed in parallel. In busy-wait operations, when the processor is idle, the computing device can serve other users through another thread.

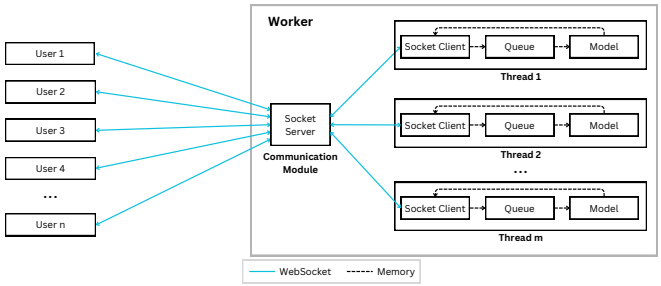

Fig. 3. Multi-threaded Data Processing via WebSocket IPC

3) Multi-process data processing via WebSocket IPC: The third method is multi-process data processing via WebSocket IPC (Figure 4). Due to the GIL issue in Python, multi-threaded structures can behave like single-threaded ones. Therefore, this method was developed to achieve higher performance. As in the multi-threaded method, data comes from the socket server via WebSocket. The key difference is that threads are replaced by processes. This approach aims

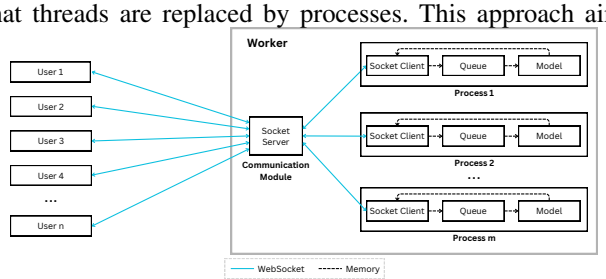

Fig. 4. Multi-process Data Processing via WebSocket IPC.

to take better advantage of the processor's computational capacity by allowing processes to run on processor cores.

4) Multi-process data processing via RESTful API: The fourth method is multi-process data processing via RESTful API (Figure 5). This method, like the third method, uses a multi-process concurrency approach but employs a RESTful API for IPC. Each process contains its own RESTful API server. When a computation request arrives, the socket server sends the data to the RESTful API using the HTTP method. Each incoming data request begins the execution process. The HTTP connection remains open throughout the computation, and the response is awaited by the socket server. With this method, queue delays have been reduced to zero.

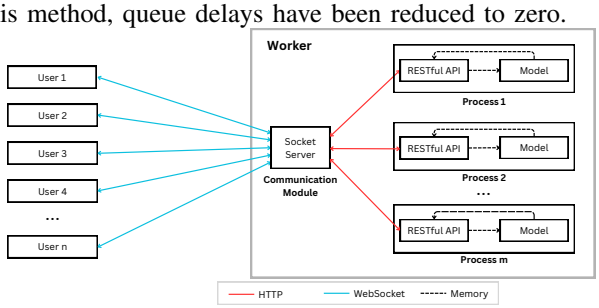

Fig. 5. Multi-process Data Processing via RESTful API.

5) Multi-process data processing via FIFO IPC: The fifth method is multi-process data processing via FIFO IPC (Figure 6). In this method, data between the socket server and the computing module is transferred using a special IPC method called FIFO. FIFO IPC, also known as the pipe method, involves two processes, one acting as the sender and the other as the receiver. Therefore, two FIFOs are required for full-duplex communication between the processes.

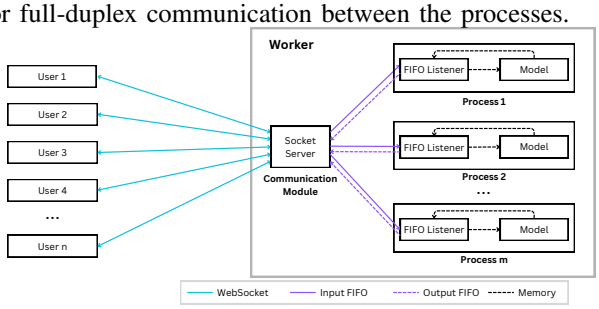

Fig. 6. Multi-process Data Processing via FIFO IPC.

The input FIFO facilitates communication from the socket server to the FIFO listener, while the output FIFO handles communication in the reverse direction.

Each time new data arrives, the FIFO listener sends the computation request to the model and does not read new data until the computation is complete. In this way, the FIFO structure also serves as a hidden queue.

Data is transmitted to processes through memory, making this method expected to be more efficient compared to other approaches. Additionally, running operations in different processes allows better utilization of multi-core processors.

The selected concurrency and IPC methods aim to fully utilize the multi-core structure of the fog device and reduce latency under multi-user load. Because the Python GIL prevents true parallel execution in multi-threading, the evaluation focused on multi-process designs. IPC variations were applied only to multi-process configurations, where inter-process communication overhead becomes meaningful.

III. Experiments and Results

In this section, the concurrency control techniques explained in Section II-C were utilized to enhance the performance of the FogETex framework. The system was first

tested with varying numbers of users for each concurrency control technique. The method that achieved the lowest response time at the highest user count was then used in the multi-worker test, and the system's performance was compared against the cloud-based implementation.

A. Experimental Setup

The test setup of the fog computing system developed for soft robotic applications consists of the following devices and configurations, as illustrated in Figure 7.

① **Traffic generator and receiver:** Dell Inspiron 15 5000 (Intel Core i7-8550U CPU @1.80GHz, 32GB DDR4 RAM, 240GB SSD, and Windows 11 Enterprise 64-bit).
② **Worker device:** Raspberry Pi 4 Model B (Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC CPU @1.8GHz, 8GB SDRAM, 32GB Micro SDHC C10).
③ **Access Point (AP):** Huwai HG8245X6 Fiber Optic Modem with ZYXEL GS-1100-16 GIGABIT Switch (ISP: Superonline, download/upload speed: 500Mbps/20Mbps).
④ **Cloud device:** Digital Ocean VPS (2x Intel CPU, 8GB RAM, 50GB SSD, Location: Frankfurt/Germany).

The traffic generator and receiver ① operate as replicas of the sensing T-IoT gloves. In the system tests, the number of users is gradually increased, and data are transmitted simultaneously to the worker devices ②. For each user, data are sent at a sampling frequency of 50 Hz. The worker device is connected to the Access Point (AP) ③, while the traffic generator and receiver access the worker devices through the AP (Wi-Fi Test Bed). A cloud device ④ is employed for benchmarking the fog system.

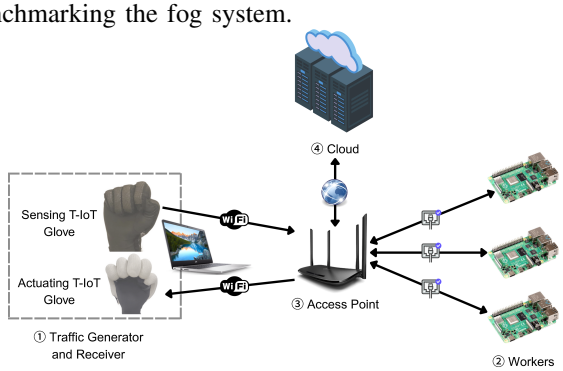

Fig. 7. Experimental setup device connection.

B. Experiments

The studies in this section were conducted using the mock client in the provided Wi-Fi Test Bed. The traffic generator and receiver emulate patients in a rehabilitation service, nursing home, or hospital. Each experiment was repeated six times, using identical data sets previously collected from a healthy test subject [22], with each repetition lasting three minutes. The model used in the tests was developed with the decision tree method, which demonstrated enhanced accuracy, resource usage, execution time, and response time compared to others [22]. The limit of how many users the devices can serve is defined as the maximum number of users that maintain an average latency of 50 ms or less for worker devices and 80 ms or less for cloud devices. These thresholds were chosen based on the jitter levels observed during preliminary tests and commonly reported limits for real-time wearable interaction. Cloud communication introduces higher network variability, so a slightly higher threshold is acceptable, whereas fog nodes must remain below 50 ms to ensure stable closed-loop control. Beyond these average response time values, the devices fail to provide real-time service, and it has been observed that the response time continuously increases with successive requests.

C. Results

In this section, the performance of different concurrency methods was evaluated in terms of response time for varying numbers of users. Based on the results, the method achieving the best performance was selected and compared with the cloud system in multi-user and multi-worker scenarios.

Figure 8 presents the mean response time of the worker device using the multi-threaded data processing method. The WebSocket IPC method was employed. In this experiment, stress tests were conducted with up to 6 threads to determine the optimal number. This maximum was chosen because the worker device has 4 cores, and using a value slightly above the core count allows more comprehensive testing. The single-thread configuration also served as the single-threaded data processing method.

The tests showed that the single-thread configuration delivered the highest performance and could serve up to 5 users. While other configurations could serve up to 4 users, the 2-thread configuration showed the lowest response time. However, the multi-threaded structure performed worse due to Python's GIL. Since other cores were not actively utilized, increasing the number of threads limited performance.

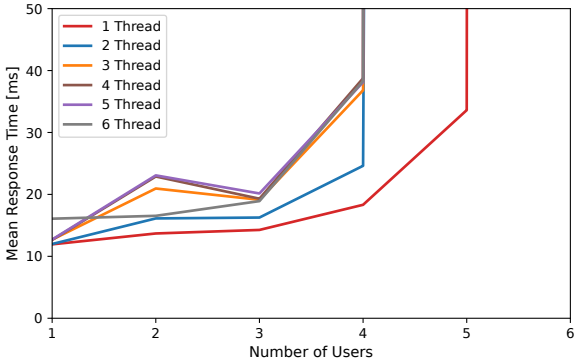

Fig. 8. Stress test for multi-threaded data processing via WebSocket IPC.

Figure 9 presents the mean response time of the worker device using the multi-process data processing method. As in the multi-threaded method, the WebSocket IPC method was employed. Different configurations with up to 6 processes were evaluated, as the worker device's 4-core processor justified using a slightly higher count.

While the single-process configuration could serve up to 5 users, the multi-process configurations were capable of serving up to 6 users. Although the results were fairly similar, the 2-process and 3-process configurations performed worse compared to the 4-process and 5-process configurations. When the number of processes was increased to 6, performance slightly decreased. The 4-process setup demonstrated

marginally better results with a lower response time than 5-process communication. This indicates that the 4-process configuration effectively utilized all the cores of the worker device's 4-core processor, leading to improved performance.

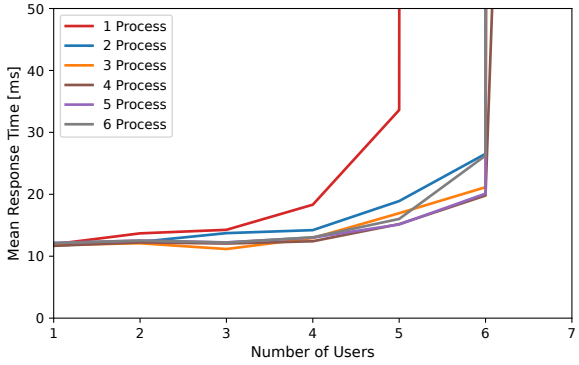

Fig. 9. Stress test for multi-process data processing via WebSocket IPC.

Figure 10 shows the mean response time of the stress test conducted using a RESTful API for IPC between the socket server and the computation module. Since the multi-process concurrency method demonstrated better performance than the multi-threaded method, this test was conducted using multi-process operations. Each computation process included its RESTful server. In tests with up to 6 users, it was observed that the single-process configuration could serve up to 3 users, while the other configurations were capable of serving up to 4 users. Among these, the 4-process configuration exhibited the best performance compared to the others.

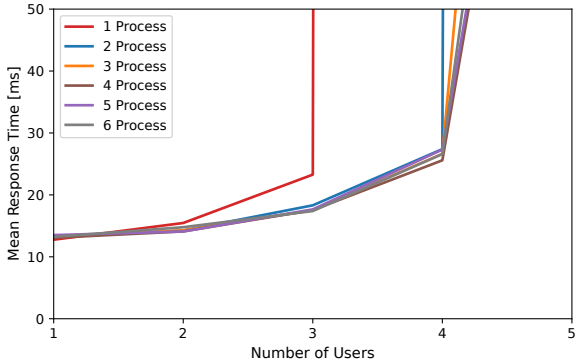

Fig. 10. Stress test for multi-process data processing via RESTful IPC.

Figure 11 presents the results of the multi-process stress test conducted using FIFO, another IPC method. In this test, an input and an output FIFO were created for each process. The test results indicate that the 2-process configuration outperformed the single-process configuration. However, increasing the number of processes beyond 2 negatively impacted performance. The single-process configuration was able to serve up to 8 users, while the 2-process configuration could serve up to 10 users.

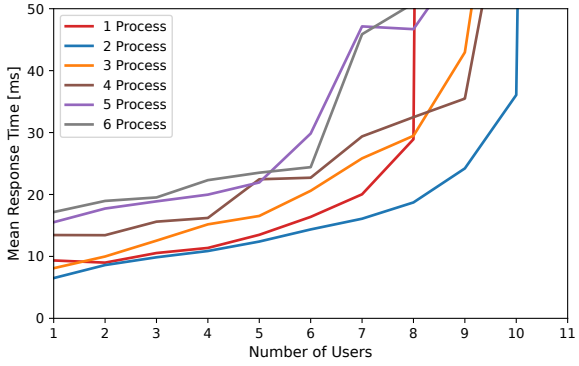

Fig. 11. Stress test for multi-process data processing via FIFO IPC.

Figure 12 compares the best-performing configurations of the proposed concurrency and IPC methods. The results show that while multi-threaded and RESTful multi-process approaches scale up to four users, the single-threaded and standard multi-process methods extend this limit to five and six users, respectively. Among all evaluated methods, the FIFO-based multi-process design achieves the highest scalability. The highest capacity, serving up to 10 users.

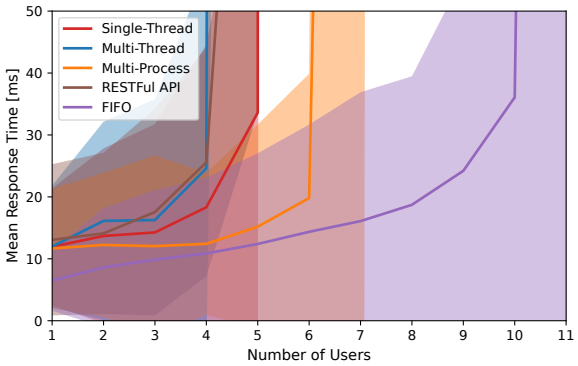

Fig. 12. Performance comparison of various concurrency techniques.

Figure 13 presents the stress test results for a fog node with multiple workers. In this test, the 2-process configuration using the FIFO IPC method, which demonstrated the best performance in the previous test, was employed. The results show that, based on mean response time, a fog node with 1 worker can serve up to 10 users, 2 workers can handle up to 22 users, and a system with 3 workers can serve up to 26 users. It was observed that the cloud system could serve up to 23 users. Therefore, to build a system with performance comparable to the cloud, it was determined that at least a 3-worker fog node configuration is required.

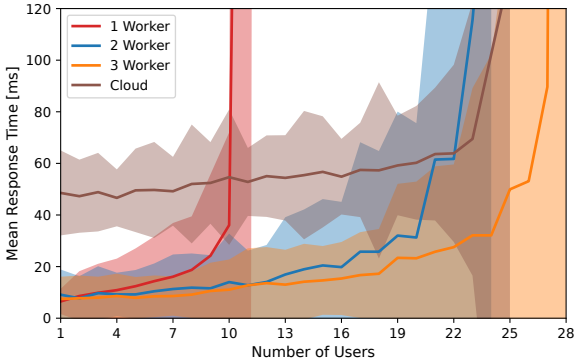

Fig. 13. Performance comparison of multi-worker and cloud.

IV. DISCUSSION

The results show that the multi-process with FIFO method achieves the best performance because it avoids GIL-related limitations and reduces IPC overhead, allowing all CPU cores to be utilized efficiently. Other methods became constrained by queue delays, thread-level blocking, or higher communication overhead, which explains their lower user

capacity. These observations clarify the performance differences among the tested approaches.

Compared with existing fog and cloud studies that rely primarily on simulations, the real-device evaluation done here shows that scalability limits appear earlier than simulations typically suggest. The ability of the proposed architecture to maintain low latency under high multi-user load demonstrates its suitability for real-time soft robotic applications.

Although the experiments were conducted with a T-IoT glove, the architecture is not application-specific and can be applied to other wearable or soft robotic systems with similar latency requirements. The evaluation of both effective and less effective configurations provides insight into multi-core utilization and communication bottlenecks, strengthening the generalizability of the findings.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose to use different concurrency and IPC methods to increase the number of concurrent users the system can handle. These methods are designed to enable soft robotic applications to serve more users efficiently. This approach facilitates the implementation of assistive systems for simultaneous operations using a mirroring technique, fully integrating textile-based sensing and actuating gloves to improve patients' quality of daily life.

The fog system was tested with different concurrency and IPC methods. The multi-process concurrency method combined with the FIFO IPC technique demonstrated the best performance. The system could serve up to 10 users with 1 worker, 22 users with 2 workers, 26 users with 3 workers, and 23 users in the cloud configuration.

In future work, the developed system can be employed to control multi-joint movements and can be adapted to non-textile soft robotic platforms. Clinical trials will be conducted to validate the system's performance. Additionally, detailed experimental validation will be carried out, including evaluations of task success rates, stability, usability, and robustness under varying network conditions, as well as user interface and safety considerations. In addition, the system can be evaluated with alternative machine learning and signal processing methods that impose higher computational demands.

ACKNOWLEDGMENT

We would like to thank Digital Ocean for providing cloud infrastructure and GitHub for repository hosting. We also acknowledge the contributions of Cagatay Gumus and Ayse Feyza Yilmaz to the application development.

REFERENCES

[1] G. M. Whitesides, "Soft robotics," *Angewandte Chemie International Edition*, vol. 57, no. 16, pp. 4258–4273, 2018.

[2] O. Yasa, Y. Toshimitsu, M. Y. Michelis, L. S. Jones, M. Filippi, T. Buchner, and R. K. Katzschmann, "An overview of soft robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 6, pp. 1–29, 2023.

[3] N. Gariya and P. Kumar, "A review on soft materials utilized for the manufacturing of soft robots," *Materials Today: Proceedings*, vol. 46, pp. 11 177–11 181, 2021.

[4] C. T. O'Neill, C. M. McCann, C. J. Hohimer, K. Bertoldi, and C. J. Walsh, "Unfolding textile-based pneumatic actuators for wearable applications," *Soft Robotics*, vol. 9, no. 1, pp. 163–172, 2022, pMID: 33481682.

[5] O. Atalay, K. Ozlem, C. Gumus, I. A. K. Ahmed, A. F. Yilmaz, M. F. Celebi, M. S. Cetin, B. Taherkhani, A. Tuncay Atalay, and G. Ince, "Thermally driven 3d seamless textile actuators for soft robotic applications," *Advanced Intelligent Systems*, vol. 6, no. 11, p. 2400133, 2024.

[6] V. Sanchez, C. J. Walsh, and R. J. Wood, "Textile technology for soft robotic and autonomous garments," *Advanced Functional Materials*, vol. 31, no. 6, p. 2008278, 2021.

[7] J. Wang and A. Chortos, "Control strategies for soft robot systems," *Advanced Intelligent Systems*, vol. 4, no. 5, p. 2100165, 2022.

[8] P. Gupta, D. K. Saini, P. Rawat, and K. Zia, *Bio-Inspired Optimization in Fog and Edge Computing Environments: Principles, Algorithms, and Systems*. CRC Press, 2023.

[9] K. C. Serdaroglu, S. Baydere, B. Saovapakhiran, and C. Charnsripinyo, "Location aware fog computing based air quality monitoring system," in *2023 International Conference on Smart Applications, Communications and Networking (SmartNets)*, 2023, pp. 1–6.

[10] A. H. Sodhro, G. H. Sodhro, M. Guizani, S. Pirbhulal, and A. Boukerche, "Ai-enabled reliable channel modeling architecture for fog computing vehicular networks," *IEEE Wireless Communications*, vol. 27, no. 2, pp. 14–21, 2020.

[11] S. S. Gill, P. Garraghan, and R. Buyya, "Router: Fog enabled cloud based intelligent resource management approach for smart home iot devices," *Journal of Systems and Software*, vol. 154, pp. 125–138, 2019.

[12] M. Kamruzzaman, S. Alanazi, M. Alruwaili, I. Alrashdi, Y. Alhwaiti, and N. Alshammari, "Fuzzy-assisted machine learning framework for the fog-computing system in remote healthcare monitoring," *Measurement*, vol. 195, p. 111085, 2022.

[13] K. Ozlem, A. T. Atalay, O. Atalay, and G. Ince, "Fogetex: Fog computing framework for electronic textile applications," *IEEE Internet of Things Journal*, vol. 12, no. 6, pp. 6856–6874, 2025.

[14] P.-H. Tsai, H.-J. Hong, A.-C. Cheng, and C.-H. Hsu, "Distributed analytics in fog computing platforms using tensorflow and kubernetes," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2017, pp. 145–150.

[15] S. H. Lee, "Real-time edge computing on multi-processes and multi-threading architectures for deep learning applications," *Microprocessors and Microsystems*, vol. 92, p. 104554, 2022.

[16] G. Galante and R. da Rosa Righi, "Adaptive parallel applications: from shared memory architectures to fog computing (2002–2022)," *Cluster Computing*, vol. 25, no. 6, pp. 4439–4461, 2022.

[17] S. Tuli, R. Mahmud, S. Tuli, and R. Buyya, "Fogbus: A blockchain-based lightweight framework for edge and fog computing," *Journal of Systems and Software*, vol. 154, pp. 22–36, 2019.

[18] A. Lebre, J. Pastor, A. Simonet, and F. Desprez, "Revising openstack to operate fog/edge computing infrastructures," in *2017 IEEE International Conference on Cloud Engineering (IC2E)*, 2017, pp. 138–148.

[19] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[20] I. Lera, C. Guerrero, and C. Juiz, "Yafs: A simulator for iot scenarios in fog computing," *IEEE Access*, vol. 7, pp. 91 745–91 758, 2019.

[21] P. Wiesner and L. Thamsen, "Leaf: Simulating large energy-aware fog computing environments," in *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*, 2021, pp. 29–36.

[22] K. Ozlem, C. Gumus, A. F. Yilmaz, A. Tuncay Atalay, O. Atalay, and G. Ince, "Cloud-based control system with sensing and actuating textile-based iot gloves for telerehabilitation applications," *Advanced Intelligent Systems*, vol. 7, no. 8, p. 2400894, 2025.

[23] H. M. Elmoughni, A. F. Yilmaz, K. Ozlem, F. Khalilbayli, L. Cappello, A. Tuncay Atalay, G. Ince, and O. Atalay, "Machine-knitted seamless pneumatic actuators for soft robotics: Design, fabrication, and characterization," *Actuators*, vol. 10, no. 5, 2021.

[24] W. Ruys, H. Lee, B. You, S. Talati, J. Park, J. Almgren-Bell, Y. Yan, M. Fernando, G. Biros, M. Erez, M. Burtscher, C. J. Rossbach, K. Pingali, and M. Gligoric, "A deep dive into task-based parallelism in python," in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2024, pp. 1147–1149.