International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(RESEARCH ARTICLE)

Check for updates

EDRA: A hybrid architecture for scalable and real-time AI applications

Shreyam Dutta Gupta *

SAP Labs, Palo Alto, California, USA.

International Journal of Science and Research Archive, 2024, 13(02), 3724-3734

Publication history: Received on 10 October 2024; revised on 26 December 2024; accepted on 28 December 2024

Article DOI: https://doi.org/10.30574/ijsra.2024.13.2.2611

Abstract

Scalability, low latency, and fault tolerance are critical requirements for modern software systems, especially for AI-powered applications like chatbots and real-time fraud detection. Modern systems often fail to balance real-time processing with scalability, leading to bottlenecks and poor fault tolerance. Traditional architectures, such as Request-Driven (RD) and Event-Driven (ED) models, provide limited scalability and fault tolerance under dynamic workloads.

To address these limitations, this paper evaluates a widely used hybrid model, Event-Driven Request Architecture (EDRA), that integrates the synchronous response handling of RD with the asynchronous scalability of ED. This study uses Python's SimPy framework to simulate EDRA and analyze its performance under varying workloads and failure scenarios. Metrics such as throughput, resource utilization, and latency are measured.

Results demonstrate that EDRA achieves approximately 20 tasks per second with 40–50% resource utilization, outperforming RD systems and matching the scalability of ED systems. Additionally, EDRA handles failures effectively using a retry mechanism, ensuring reliability for up to 8 consumers.

Our findings highlight that EDRA balances real-time responsiveness with scalable background processing, making it a viable architecture for AI-driven and high-load applications. This study provides a scalable framework applicable to AI systems, e-commerce platforms, and IoT devices. It also offers actionable insights for developers and system architects designing scalable, fault-tolerant systems.

Keywords: Event-Driven Architecture; Request-Driven Architecture; Hybrid Architecture; Microservices; Scalability; Fault Tolerance; Real-Time Systems

1. Introduction

Modern software systems require scalability, fault tolerance, and real-time responsiveness to support AI-powered applications such as chatbots and fraud detection systems. Traditional architectures—Request-Driven (RD) and Event-Driven (ED)—partially address these requirements but often fall short in balancing synchronous and asynchronous workloads, leading to bottlenecks and reduced reliability.

RD architectures, commonly used for CRUD operations and immediate feedback, rely on synchronous processing. Asynchronous techniques, such as long polling and messaging queues (e.g., Apache Kafka), can extend RD architectures [1]. However, they still struggle under high loads due to tight coupling and limited scalability. In contrast, ED architectures excel at modularity and fault isolation by enabling asynchronous, decoupled communication through event-driven workflows. Despite these advantages, ED systems introduce complexities, such as managing consistency, handling failures, and coordinating distributed processes [2].

* Corresponding author: Shreyam Dutta Gupta

To address these challenges, this paper adopts the term Event-Driven Request Architecture (EDRA) to describe a widely used hybrid approach that integrates RD's synchronous response handling with ED's asynchronous scalability and fault tolerance. While this model is not new, it has lacked formal recognition in prior literature.

The objective of this study is to evaluate EDRA's performance under dynamic workloads, focusing on scalability, reliability, and fault tolerance. Using Python's SimPy framework, we simulate EDRA and analyze metrics such as throughput, resource utilization, and latency. Simulation provides a controlled environment to model real-world scenarios, analyze performance under varying workloads, and test fault tolerance without requiring costly deployment of large-scale infrastructure. This approach allows us to validate EDRA's ability to handle high-load applications and failure scenarios effectively.

Our findings demonstrate that EDRA effectively balances real-time responsiveness with scalable background processing, offering actionable insights for developers and system architects building fault-tolerant, high-load systems.

1.1. Related work

Prior studies have analyzed SOA and EDA as related paradigms with a focus on the challenges of developing tightly coupled systems in SOA [3]. Clark and Barn [4] in their view consider EDA to be a specialty of SOA where event-based interaction replaces the typical request response model, thus increasing flexibility and scalability. Their work highlights that EDA can enable Complex Event Processing (CEP), because, in contrast to SOA, which is often synchronous, it can address both synchronous and asynchronous events across systems. The idea of using both SOA and EDA is consistent with the Event-Driven Request Architecture (EDRA) proposed in this paper for real time responses and scalable background processing of tasks.

The concept of adaptive architecture has been discussed in different situations to enhance scalability and performance. For example, Azzedin and Al-Issa [5] proposed a self-adaptive web server architecture that uses the technique of synchronous and asynchronous I/O operations depending on the actual load. This improvement creates positive outcomes in terms of optimizing the activity of the system by allowing it to adjust in response to the system load. Similarly, the Event-Driven Request Architecture (EDRA) makes use of the flexibility of the synchronous and asynchronous layering for handling real-time user interactions and at the same time delivers without the interruption from synchronous traffic while handling high volumes of background workloads. Hence, both architectures propose to make the best usage of resources and scaling, depending on the varying traffic demands.

The studies done by Carrera et al. [6] in modeling web servers have revealed how the application of Multithreaded and Event-driven hybrid architectures could offer better performance. These approaches especially in e-commerce applications focus on how efficiently sessions can be managed across varying load—a concept enhanced using EDRA for processing real time requests and other event-driven processes in Artificial Intelligence and its applications, which we'll validate using a simulation study.

1.2. Hybrid model: event driven request architecture

The Event-Driven Request Architecture (EDRA) combines the Request Driven architecture's immediate feedback and Event Driven architecture's scalability and fault tolerance. Using such a hybrid model offers substantial benefits to the applications that require real time processing but also need background processing for asynchronous tasks.

1.2.1. Key principles

- Real-time synchronous requests for critical interactions: User facing operations such as queries to a chatbot or any search functionality, are performed synchronously using a Request Driven approach to minimize response time. If the application needs streaming of data, concepts like Long Polling discussed previously can be used.
- Asynchronous event handling for background processing: Event Driven principles are implemented in the processes that do not need to be completed in real time like analytics or triggering batch jobs or notification service.
- Decoupling of subsystems: Subsystems are loosely coupled and communicate by passing event messages where response is not expected in real time. However, the subsystems that require synchronous communication directly connected for the faster processing of the requests and immediate response.

1.3. Theoritical foundation

In this section, we describe formal models for Request Driven, Event Driven, and Event Driven Request Architecture to analyze their performance characteristics quantitatively.

1.3.1. Request-Driven Architecture (RD)

Queuing Model

M/M/1 queue (single-server queue with Poisson arrivals and exponential service times) [7].

Parameters

- Arrival Rate (λ): Mean number of requests arriving per unit time.
- Service Rate (μ): Mean number of requests the server can process per unit time.
- Utilization Factor (ρ): Representing the fraction of time the server is busy.

$$\rho = \frac{\lambda}{\mu}$$

Reasoning: In real-time systems like fraud detection, the M/M/1 model represents how the system processes a single event/transaction in real-time. The arrival rate λ indicates the number of fraud-check requests and the service rate μ represents how quickly the AI model can return a decision. This paradigm applies to chatbots when instant responses are required for example in Q&A systems where users demand quick answers.

Performance Metrics:

- Average Number of Requests in the System (Little's law) (L)

$$L = \frac{\rho}{1-\rho}$$

- Average Time a Request Spends in the System (W)

$$W = \frac{1}{\mu - \lambda}$$

- Average Time a Request Spends Waiting (W_q):

$$W_q = \frac{\rho}{\mu - \lambda}$$

Analysis

- Low Load Conditions ($\lambda \ll \mu$): The system responds quickly with minimal waiting times.
- High Load Conditions ($\lambda \to \mu$): The average waiting time and number of requests in the system increase exponentially, leading to potential bottlenecks.

1.4. Event-Driven Architecture (ED)

1.4.1. Queuing Model

M/M/c queue (multi-server queue with Poisson arrivals and exponential service times) [7].

Parameters:

- Arrival Rate (λ_e): Mean number of events arriving per unit time.
- Service Rate (μ_e): Mean number of requests the server can process per unit time.
- Number of Consumers (c)

- Utilization per Consumer (ρ_e):

$$\rho_e = \frac{\lambda_e}{c\mu_e}$$

Reasoning: The M/M/c model is appropriate for asynchronous tasks where multiple events are processed in asynchronously. For example, in a chatbot which uses natural language processing (NLP) can asynchronously handle multiple user conversations at once, where each conversation is treated as an independent task handled by separate consumers. In real time fraud detection systems, batch model updates or analysis of large datasets can also be processed asynchronously using this queuing model.

Performance Metrics

- Probability that All Consumers are Busy (P_{wait}): Calculated using Erlang's C formula.
- Average Number of Events in the System (L_e):

$$L_e = c\rho_e + \frac{\rho_e^{c+1}}{c!\,(1-\rho_e)^2}\left(\frac{c\mu_e}{\lambda_e}\right)^2$$

- Average Waiting Time in Queue (W_{qe}):

$$W_{qe} = \frac{L_e}{\lambda_e} - \frac{1}{\mu_e}$$

Analysis

- Scalability: By increasing c, the system can handle higher arrival rates with acceptable waiting times.
- Resource Utilization: Proper sizing of c is critical to balance utilization and service levels.

1.5. Event-Driven Request Architecture (EDRA)

1.5.1. Combined Queuing Model

Synchronous Layer (RD): M/M/1 queue.

- Arrival Rate (λ_s): Mean number of requests arriving per unit time.
- Service Rate (μ_s): Mean number of requests the server can process per unit time.
- Utilization ($\rho_e = \frac{\lambda_s}{\mu_s}$): Representing the fraction of time the server is busy.

Asynchronous Layer (ED): M/M/c queue.

- Event Arrival Rate (λ_a):

$$\lambda_a = p\lambda_s$$

where p is the probability that a synchronous request generates an event.

- Service Rate per Consumer (μ_a)
- Number of Consumers (c)
- Utilization per Consumer:

$$\rho_a = \frac{\lambda_a}{c\mu_a}$$

Reasoning: In AI applications, employing Event-Driven Request Architecture (EDRA) balances real-time interactions with asynchronous processing. If we take chatbot as an example, the synchronous layer (M/M/1) processes the user interactions in real time, while ensuring immediate feedback and the asynchronous layer (M/M/c) handles tasks like sentiment analysis or personalized recommendations, without interrupting the user queries. The system's scalability can be enhanced by adding more consumers (c) to the asynchronous layer, which will allow it to handle the increasing event loads, such as multiple concurrent user sessions in a chatbot. This architecture is also fault tolerant, since failures in the asynchronous layer do not affect the real-time processing of critical tasks in the synchronous layer.

Performance Metrics

Synchronous Layer Metrics

- Average Response Time (W_s)

$$W_s = \frac{1}{\mu_s - \lambda_s}$$

- Throughput (X_s):

$$X_s = \lambda_s$$

Asynchronous Layer Metrics

- Average Waiting Time (W_{qa}): Calculated using Erlang's C formula for the M/M/c queue.
- Average Response Time (W_a):

$$W_a = W_{qa} + \frac{1}{\mu_a}$$

- Throughput (X_a):

$$X_a = \lambda_a$$

Total System Metric

- Combined Throughput (X_{total}):

$$X_{total} = X_s + X_a$$

- Total Average Response Time (W_{total}):

$$W_{total} = W_s + pW_{qa}$$

Analysis

- Latency: Critical tasks will run at a near-zero latency on the synchronous layer while the asynchronous layer does the same for non-critical tasks without affecting the synchronous layer's performance of its key tasks.
- Scalability: The System will be horizontally scalable by increasing the number of asynchronous consumers (c) to handle increasing event loads.
- Fault Tolerance: Failures in the asynchronous layer are isolated from the synchronous layer, enhancing overall system reliability

2. Implementation and metrics

2.1. Simulation Setup

Tool: Python, SimPy (discrete-event simulation) [8].

2.1.1. Simulation Parameters:

- Simulation Time (T): 10,000 seconds.
- Service Rates:
  - Synchronous Server (μ_s): 25 requests per second.
  - Asynchronous Consumers (μ_a): 25 events per second.
- Arrival Rates (λ): From 1 to 20 requests per second.
- Number of Consumers (c): 1, 2, 4, and 8.
- Event Generation Probability (p): 0.5.
- Failure Rate (f): 10% chance of failure in the asynchronous layer.
- Maximum Retries (r_{max}): 3 retries allowed for failed events.

Simulation Algorithms

RD Simulation Algorithm

Initialize the simulation environment and create a server resource with capacity 1.

Request Generator

- Generates requests based on the arrival rate (λ).
- Each request:
  - Waits for inter-arrival time (exponentially distributed).
  - Requests access to the server.
  - Upon access, processes synchronously for a service time (exponentially distributed).
  - Releases the server.
  - Records latency and updates throughput and utilization metrics.

ED Simulation Algorithm

Initialize the simulation environment and create a consumer resource with capacity c.

Event Generator

- Generates events based on the arrival rate (λ).
- Each event:
  - Waits for inter-arrival time.
  - Requests access to a consumer.
  - Upon access, processes asynchronously for a service time.
  - Releases the consumer.
  - Records latency and updates throughput and utilization metrics.

EDRA Simulation Algorithm

Initialize the simulation environment with a synchronous server and asynchronous consumers.

Request Generator

- Generates requests based on the arrival rate (λ).
- Each request:
  - Processes synchronously as in the RD model.
  - With probability p, generates an asynchronous event.

Asynchronous Event Handler:

- Each event:
  - Requests access to a consumer.
  - Upon access, processes asynchronously.
  - Simulates failure based on the failure rate (f).

If failed:

- Retries up to r_{max} times.
- If maximum retries exceeded, records as a failed event.
  - Releases the consumer.
  - Records latency and updates throughput and utilization metrics.

Metrics Calculation:

- Throughput: Calculated by dividing the number of completed tasks by the simulation time.
- Utilization: Calculated by dividing the total busy time of resources by the simulation time.
- Latency: Averaged over all completed tasks.

Failed Events: Counted during the failure injection analysis.

3. Results

3.1. Throughput vs. Arrival Rate

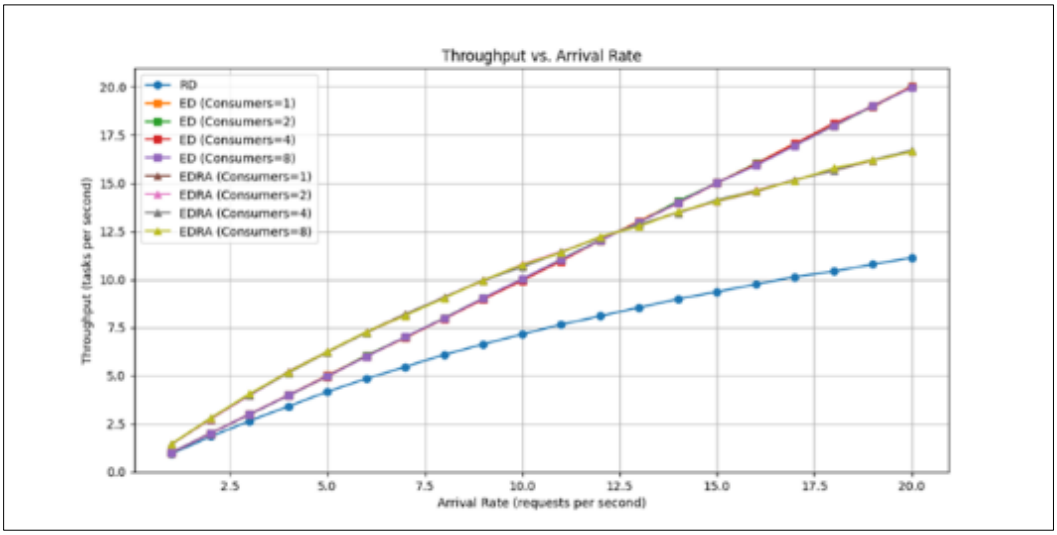

Figure 1 Throughput vs. Arrival Rate

3.1.1. Observations

- Request Driven (RD) systems have the lowest throughput across all arrival rates, suggesting scalability limitations as load increases. This is typically expected because a single server would handle all requests, resulting in bottlenecks.
- Event Driven (ED) increases throughput dramatically (from one to eight users). In this architecture, the load is handled considerably more efficiently in systems with more customers (ED with 8), and the arrival rate approaches linear scaling as it grows.
- However, we show that the hybrid EDRA model, which includes more consumers, scales well and meets (or exceeds) RD throughput. Nonetheless, with the same number of clients, EDRA has a somewhat lower throughput than the ED system because it requires more synchronization overhead to process requests.

3.1.2. Insights

- Both ED and EDRA models scale better than RD as arrival rates increase.
- ED/EDRA leads to better performance, and the highest throughput is obtained in both models with 8 consumers.

3.2. Resource Utilization vs. Arrival Rate

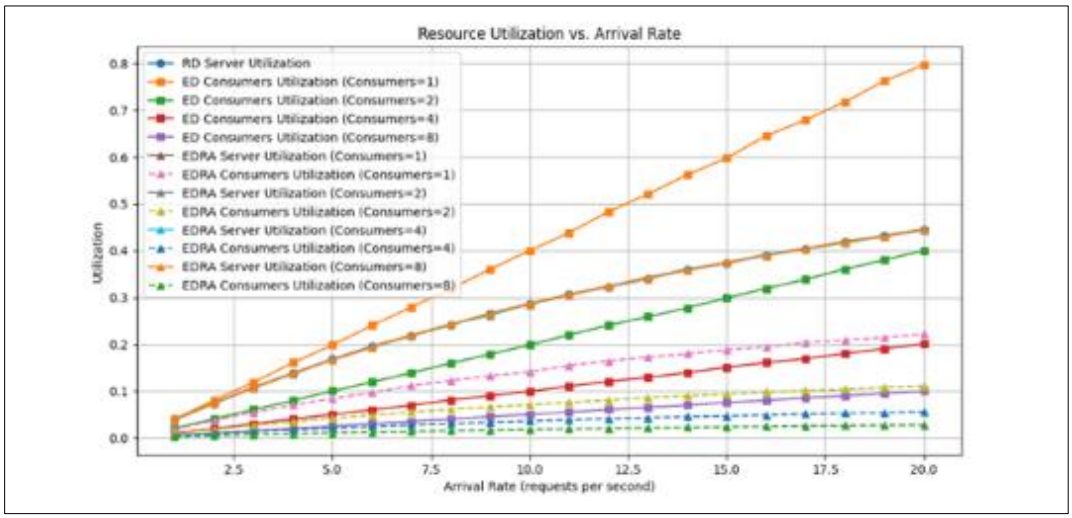

Figure 2 Resource Utilization vs. Arrival Rate

3.2.1. Observations

- As the arrival rate increases, the RD server's usage gradually increases, peaking at around 0.6 to 0.7. This is due to the synchronous handling of all requests, which causes bottlenecks.
- Event-driven consumers in ED systems with fewer consumers i.e. 1 or 2 have significantly higher utilization compared to those with more i.e. 4 or 8. This is because higher number of consumers can more efficiently distribute the load. Utilization increases steadily as the arrival rate rises.
- EDRA reduces server utilization by offloading work to asynchronous consumers, compared to RD. With more users, usage remains well below saturation, demonstrating the hybrid model's capacity to spread the load.
- In EDRA, like ED, higher arrival rates lead to increased consumer utilization, but this is dispersed across consumers. Higher consumer counts result in reduced utilization per customer, demonstrating effective load distribution.

3.2.2. Insights

- High arrival rates lead to higher utilization and bottlenecks in RD.
- ED and EDRA efficiently handle more consumers, reduces bottlenecks and manages utilization.

3.3. Failed Events vs. Arrival Rate in EDRA with Failure Injection

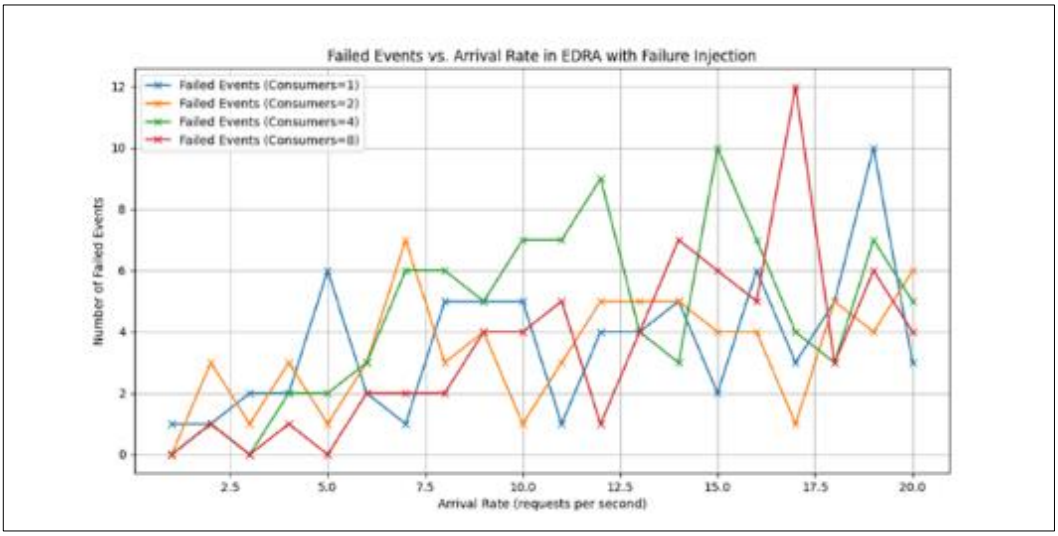

Figure 3 Failed Events vs. Arrival Rate in EDRA with Failure Injection

3.3.1. Observations

- As arrival rates increase, the number of failed events fluctuates.
- General trends show modest failure rates (usually below 5), although spikes occur as arrival rates grow.
- In systems with more consumers, there are fewer failed events overall, but there are anomalies. For example, when the arrival rate ranges from 17 to 18 requests per second, the system with 8 consumers has a significant rise in failed events.
- The frequency of failed events is normally larger in systems with fewer customers (e.g., one consumer), but especially when arrival rates are high.

3.3.2. Insights

- With more asynchronous consumers, the system can handle asynchronous events more efficiently, leading to fewer failures overall.
- The sharp spikes suggests that when the system becomes overloaded, it results in retries failing even with multiple consumers.

3.4. Average Latency vs. Arrival Rate with Failure Injection

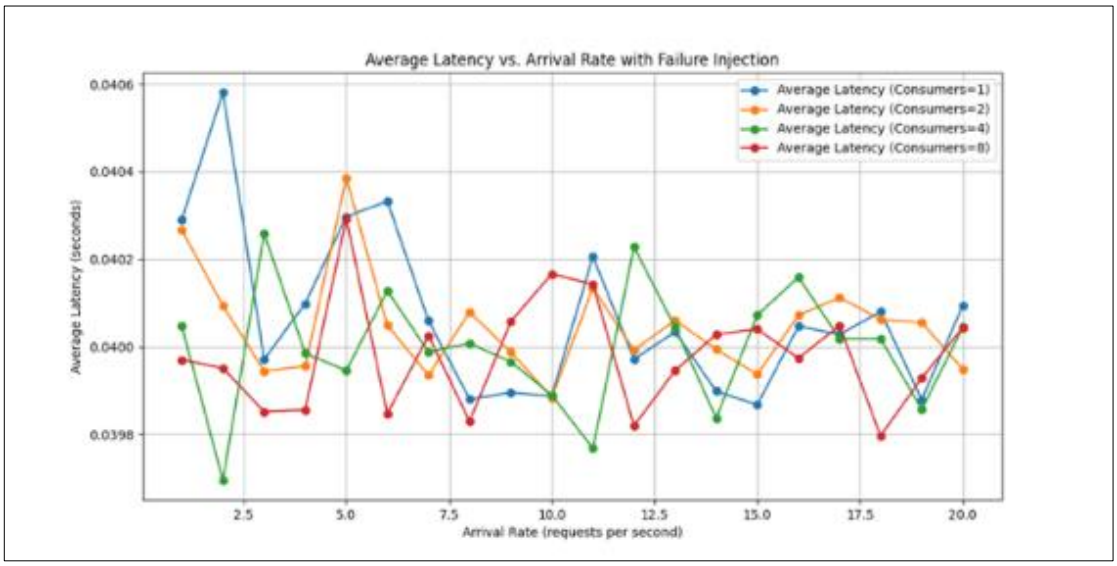

Figure 4 Average Latency vs. Arrival Rate with Failure Injection

3.4.1. Observations

- The average latency remains within a narrow range (0.0398 to 0.0406 seconds) independent of consumer count or arrival rate.
- At lower arrival rates e.g., around 5 requests per second, minor fluctuations are visible, but they have a minimal impact on average latency.
- Asynchronous events do not significantly impact latency, even when the arrival rate grows.

3.4.2. Insights

- Since the system can handle events quickly, it doesn't let the number of consumers or arrival rate significantly influence latency. This implies that errors are quickly attempted again, reducing their effect on the total amount of time needed to process requests.
- Higher consumer counts help maintain a lower latency as arrival rates grow, although the growth is not significant owing to efficient processing.

4. Comparative analysis and key insights

The analysis reveals that Event-Driven Request Architecture (EDRA), and Event-Driven (ED) systems consistently outperforms the Request-Driven (RD) model in terms of throughput, resource utilization, and the ability to handle higher arrival rates. Here's a complete comparison:

4.1. Throughput Comparison

- **Request-Driven (RD):** In terms of throughput, RD performs the worst. It grows linearly and the system becomes a bottleneck because the server will process the requests sequentially. When the arrival rate reaches the upper limit, it maxes out at ~10 tasks per second, indicating that the single server cannot handle the demand.
- **Event-Driven (ED):** ED systems scale much better than RD systems due to the parallelism from multiple consumers. At high arrival rates, ED can serve ~ 20 tasks per second with 8 consumers, which is twice more than RD. The throughput is efficient and grows nearly linearly with the arrival rate.
- **Event-Driven Request Architecture (EDRA):** EDRA performs similarly to ED but has a slight overhead from synchronous request handling. Like ED, the highest-performing configuration is EDRA with 8 consumers, achieving nearly 20 tasks per second, while also balancing synchronous and asynchronous requests.

4.2. Resource Utilization Comparison

- **RD**: For RD systems, at higher arrival rates the system becomes heavily utilized, hitting as high as 70%, indicating that the system is reaching its capacity.
- **ED**: For ED systems, 1 or 2 consumers results in 80% consumer utilization which might cause bottleneck, but 4 or 8 consumers makes much better load distribution, resulting in lower consumer utilization of 50-60%, which indicates a good balance between handling requests and preventing saturation.
- **EDRA**: The EDRA system shows balanced server and consumer utilization. The server utilization is kept low around 40 to 50%, thanks to the asynchronous offloading of events to consumers. Consumer utilization is kept reasonable and balanced, with higher consumer counts translating to an even distribution of load, mitigating the likelihood of creating bottlenecks.

4.3. Failure Handling:

- EDRA can handle failure injection and retry failed events, which is a benefit over pure ED. It performs well even when asynchronous events fail and retries do not dramatically reduce throughput or latency.

4.4. Best Performing Architecture and Consumer Count:

- Architecture: For best overall performance, EDRA with 8 consumers takes the lead. It balances synchronous and asynchronous requests efficiently between clients and servers. The consumer utilization does not reach saturation, and it generates the highest throughput with ~20 processes per second.
- Consumer Count: With 8 consumers, both the ED and EDRA systems performs well; however, at 4 consumers the system works fine, but with 8 consumers it will have enough leeway to provision for growing loads and failures without a bottleneck.

Simulations confirm that **EDRA with 8 consumers** is the most balanced, scalable, and fault-tolerant configuration. Its decoupling of services supports independent scaling while preserving responsiveness—critical for microservices-based architectures. This aligns with prior research [9], where Event-Driven Architectures improved response time by 19.18% and reduced error rates by 34.40%.

5. Conclusion

This study demonstrates that Event-Driven Request Architecture (EDRA) effectively balances synchronous and asynchronous processing, offering scalability, fault tolerance, and low latency. EDRA consistently outperforms traditional Request-Driven (RD) systems and matches Event-Driven (ED) scalability while addressing fault tolerance through retry mechanisms.

Simulations confirm that EDRA with 8 consumers achieves optimal performance, supporting up to 20 tasks per second with 40–50% resource utilization. Its ability to handle failures without degrading throughput or latency makes it a practical choice for AI systems, microservices, and IoT applications.

By decoupling services and enabling independent scaling, EDRA provides a flexible framework for modern distributed systems. Future work will focus on refining fault-tolerance mechanisms, extending EDRA to edge computing, and optimizing real-time event handling for globally distributed workloads.

Future work

As software complexity grows, the limitations of RD and ED become evident. EDRA addresses these gaps by combining RD's real-time responsiveness with ED's scalable, asynchronous processing. It is particularly well-suited for AI digital assistants, e-commerce, and microservices, where scalability, fault tolerance, and modularity are essential.

Simulations demonstrate EDRA's superior throughput, resource utilization, and fault tolerance, especially with optimized asynchronous consumers. With distinct services, message brokers, and fault-tolerance mechanisms, EDRA offers a scalable and flexible solution for high-volume, real-time applications.

While EDRA's hybrid nature adds some complexity, its scalability and performance make it ideal for microservices. Future work includes refining message brokers, improving real-time event handling, and incorporating AI/ML workflows for enhanced efficiency. Expanding EDRA to edge computing and globally distributed systems promises even greater scalability and resilience, shaping it as a framework for modern distributed systems.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Ionescu VM. The analysis of the performance of RabbitMQ and ActiveMQ. In2015 14th RoEduNet International Conference-Networking in Education and Research (RoEduNet NER) 2015 Sep 24 (pp. 132-137). IEEE.

[2] Vemulapalli G. Architecting for Real-Time Decision-Making: Building Scalable Event-Driven Systems. International Journal of Machine Learning and Artificial Intelligence. 2023 Feb 16;4(4):1-20.

[3] Lan L, Li F, Wang B, Zhang L, Shi R. An event-driven service-oriented architecture for the Internet of Things. In2014 Asia-Pacific Services Computing Conference 2014 Dec 4 (pp. 68-73). IEEE.

[4] Clark T, Barn BS. Event driven architecture modelling and simulation. InProceedings of 2011 IEEE 6th International Symposium on Service Oriented System (SOSE) 2011 Dec 12 (pp. 43-54). IEEE.

[5] Azzedin F, Al-Issa K. A self-adapting Web server architecture: Towards higher performance and better utilization. In2009 International Conference on High Performance Computing & Simulation 2009 Jun 21 (pp. 96-105). IEEE.

[6] Carrera D, Beltran V, Torres J, Ayguadé E. A hybrid web server architecture for e-commerce applications. In11th International Conference on Parallel and Distributed Systems (ICPADS'05) 2005 Jul 20 (Vol. 1, pp. 182-188). IEEE.

[7] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, Fundamentals of Queueing Theory, 4th ed. Hoboken, NJ: Wiley, 2011.

[8] SimPy [Internet]. Available from: https://simpy.readthedocs.io/en/latest/contents.html

[9] Rahmatulloh A, Nugraha F, Gunawan R, Darmawan I. Event-Driven Architecture to Improve Performance and Scalability in Microservices-Based Systems. In2022 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS) 2022 Nov 23 (pp. 01-06). IEEE.