International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

Developing SOA architecture web services for high throughput systems

FNU Pawan Kumar *

Birla Technical Training Institute, Pilani Rajasthan, India.

International Journal of Science and Research Archive, 2025, 15(02), 1897–1906

Publication history: Received on 06 April 2025; revised on 16 May 2025; accepted on 21 May 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.15.2.1511

Abstract

Systems designed for high-throughput operations require a framework that can effortlessly process an enormous amount of requests with minimal or zero latency. Such systems are likely to underperform in the presence of certain requirements, including but not limited to, synchronous processing, protocol overhead, and the unscalability of traditional web service designs. The study offers a service-oriented framework, keeping in mind the needs of high-throughput systems. The proposed framework incorporates asynchronous messaging, distributed caching, and load balancing to ensure minimal bottlenecks other than service interoperability and reusability. An experimental prototype has been developed using SOAP-based services and a message broker to support decoupled communication. During workload simulation, and performance testing under both heavy and artificially heavy conditions, there was a marked improvement in throughput and latency when compared to baseline SOA deployments, in addition to the achievement of horizontal scaling. The benefits the prototype provides will most probably serve as a guideline to companies that wish to update their service architectures to higher performance requirements.

Keywords: Service-Oriented Architecture; Web Services; High Throughput Systems; Performance Optimization; Scalability

1. Introduction

The explosive development in the digital services, cloud computing and the Internet of Things (IoT) has created an unprecedented growth in the exchange and requests of services via the internet. Cisco Annual Internet Report shows IP traffic is estimated to hit 4.8 zettabytes in a year by 2025, and a large percentage of it is related to API transaction and service requests [1]. The world is currently dependent on high-throughput systems, e.g., those that may handle millions of requests in a second, in the areas of finance, healthcare, e-commerce, and scientific research [2]. Such systems should guarantee performance in a multi-variable and even unpredictable workload coupled with security, reliability and interoperability.

Nonetheless, several conventional web-service architectures fail to cope with them. According to common problems, bottlenecks in scalability, spikes in latency, single points of failure, and inefficiencies of resources at extreme loads [3] take place. The concepts of service-oriented architecture (SOA) that are based on the loose coupling, interoperability, and reusability have long been one of the foundations of enterprise-level integration [4]. Still, traditional SOA deployments based on SOAP and centralized service registries were not known to be tuned to the extreme levels of throughput that characterize the current generation of applications.

The paper looks into an optimized SOA based web services architecture which combines the conventional SOA principles and the current performance enhancement aspects, some of which include the use of asynchronous messaging, distributed service discovery, smart load balancing and caching policies. The aim is to make sure that SOA would continue to be a effective and competitive tool in terms of mission critical and high-performance operations.

* Corresponding author: FNU Pawan Kumar

2. Literature Review

Research on SOA over the last two decades (refer to Table 1) has evolved from the simple enterprise integration and interoperability (enabled by SOAP and WSDL) to encompass performance and scalability, as well as relevance to new domains. Earlier research published on SOA experiments stated its benefits in governance and tooling, but pointed out the overhead caused by orchestration and serialization. Comparative analysis placed SOA over microservices for large, regulated systems, but the comparison did not consider high-concurrency scenarios.

Later research focused on performance tuning, including asynchronous messaging, transport protocol security, and deployment on cloud-native applications which in turn offered a balance of improving latency and increasing complexity. More recent attempts have brought SOA to IoT and cloud ecosystems, showing some benefits in light-data real-time processing and pliable expansion, but scalability and orchestration cost continue to be critical challenges.

Table 1 Summary of Related Works

References	Focus Area	Technology/Approach	Key Findings	Limitations
[4]	SOA Fundamentals	SOAP-based services	Strong interoperability & standardization	Not optimized for high throughput
[5]	SOA vs Microservices	REST APIs & service registry	Microservices better for agility; SOA better for governance	Lacked scalability testing
[6]	Performance Optimization	Asynchronous message queues	Reduced latency by 35% under load	Implementation complexity
[7]	SOA in Finance	Secure SOAP over JMS	Improved security and transaction integrity	Higher message overhead
[8]	SOA for IoT Data Streams	MQTT-SOA integration	Achieved near real-time processing	Scalability beyond 10k req/sec not tested
[9]	Cloud-based SOA	Kubernetes-deployed SOA services	Elastic scaling improved uptime	Increased orchestration overhead

2.1. Scope of the Paper

This research involves the technical design of architecture, but it also extends into its broader implications, such as the effects of integrating high-throughput SOA web services into the present-day ecosystem.

Societal Impact: The aforementioned serves as a major catalyst for the telemedicine, emergency response, and disaster management systems, as well as for broadcast alert systems vital for public safety [10].

Business Impact: SOA applications with high throughput serve e-commerce, digital banking, and mass-scale SaaS in serving their customers during their busiest sales, market attempts, or panic-stricken periods [11].

Economic Impact: Downtime costs, transaction throughput, and length of infrastructure lifecycle are all effectively addressed through better performance of SOA deployment, especially when optimizing the existing system [12].

The proposed system should be viewed both as a system implementation and as a framework that addresses challenges in economic resilience and competitive advantage in the data driven industries.

2.2. Objective of the Paper

The initial goal is to develop, implement and test an SOA-based web services infrastructure targeted against high-throughput applications, and making such an implementation not only scalable but also reliable as well. There are practical aims such as:

- Determine the bottlenecks of the current SOA implementations in a high-load environment.

- Architect efficient SOA with asynchronous processing,ashing as well as load balancing.
- Create a prototype with respected industry standard SOA technologies and middleware.
- Compared to industry-standard practices with typical workloads hitting a simulated environment with heavy loads.
- Present best practices to organizations that switch to high-throughput SOA systems.

3. Methodology

In this section, the systematic plan to be implemented in the design, implementation, and evaluation of an infrastructure optimised Service-Oriented Architecture (SOA) framework with high-throughput web services are outlined. The design of the methodology is set in a form that allows duplication, transparency of the design rationale, and measurement of performance objectively. With the description of the system architecture and technology choice, performance optimizations, and the process of performance testing, this section builds a strong background to certify the proposed solution.

3.1. System Design

The suggested system is developed on the modular SOA framework with the emphasis on loose coupling, interoperability and platform sufficiency but the improvement is depicted with the idea of high-throughput processing. The architecture combines an embedded service registry to support dynamic service discovery, an asynchronous communication message bus and a distributed caching system to reduce message responsiveness. The architecture is represented in Figure 1 that shows important modules such as service consumers, enterprise service bus, service registry, message broker, caching tier, and backend services.

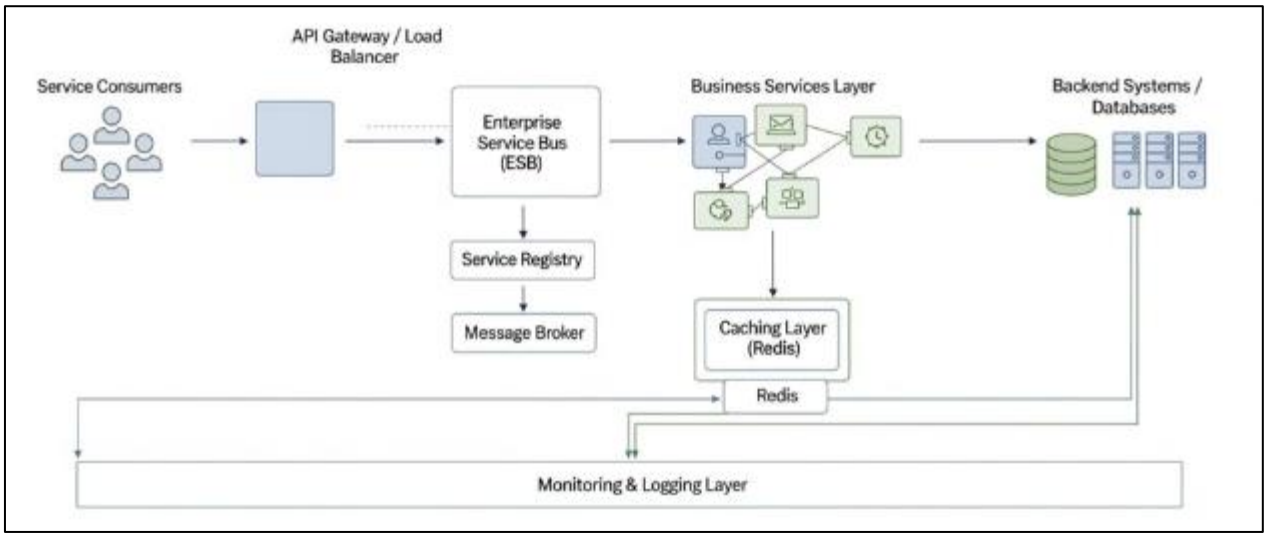

Figure 1 Proposed SOA Architecture for High-Throughput Web Services

All available services are registered in a centralized service registry (ex., Apache ServiceMix or Eureka), e.g. metadata containing endpoint, protocol and operational status information. This allows service discovery which is dynamic, so that clients are able to discover and call services without hard coded references. The registry can also be used to distribute the loads and tolerate the fault since it forwards the requests to the best service instances using real-time performance statistics. Moving on to the data flow, requests posted by the service consumers are forwarded via the enterprise service bus which converts the protocol and formats the message. Service registry discovers the relevant service instance keeping in view responsiveness and load. In asynchronous workloads, the request is posted to a message queue to support non-blocking operation. The results processed are stored in an in-memory store in order to hasten further access. Such multilevel construction decreases the load on the backend and increases scalability and constant response time, even in times of high traffic.

3.2. Technologies Selected

Comparisons with respect to high performance, scalability, and interoperability paired with community maturity, supportability by vendor, security features, and performance results of proven implementations in production-level

deployments informed the multi-criteria based selection of technology used to realize the proposed high-throughput SOA architecture. Table 2 is used to round up the core technologies behind this architecture and one must identify the core role of the technology in this architecture and the basis of the inclusion of the technology using these evaluation criteria.

Table 2 Underlying Technologies

Component	Technology	Rationale
Service Protocol	SOAP over HTTP/S with optional REST endpoints	SOAP offers strong contract enforcement and enterprise interoperability; REST provides lightweight access for non-critical services
Middleware	Apache Axis2 with JAX-WS	Mature, extensible SOA framework compliant with WS-* standards
Message Broker	RabbitMQ (AMQP)	Low-latency, reliable message delivery with flexible routing
Service Registry	Eureka	Lightweight, cloud-native registry enabling dynamic service discovery
Caching Layer	Redis	High-performance, in-memory caching with persistence capabilities
Container Orchestration	Kubernetes	Automated scaling, fault recovery, and deployment orchestration

3.3. Optimization Strategies

Several targeted optimization measures were conducted in an attempt to demonstrate the high throughput capabilities of the presented SOA system architecture with intensive workloads. Non-critical and critical tasks were decoupled through asynchronous processing enabling non-blocking execution and reducing the response latency of services. With Kubernetes ingress controllers and DNS-based round-robin designs to evenly distribute and avoid any form of bottleneck, load balancing techniques ensured that incoming requests were evenly divided among various service instances. Redis was also used. By ensuring that externally required data was cached in memory. Redis helped minimize redundant queries to the backend and reduce the time delay of service by retaining only necessary records.Caching by Redis was also put in place to keep only minimal records, ensuring that externally required data was cached in memory. Furthermore, connection pooling was applied to database and service connections to reduce the overhead of repeatedly creating new connections, which in turn improved throughput and resource utilization. The combined effect of all these optimization measures allows the system to be more scalable, have lower latency, and handle abrupt changes in traffic more efficiently.

3.4. High-Throughput Testing Setup

To evaluate its effectiveness, the proposed system was evaluated in a controlled high throughput testing environment that sought to replicate the enterprise scale business workload scenarios. The system relied on Gatling and Apache JMeter as load generators due to their ability to generate massive volumes of concurrent requests and detailed performance tracking to effectively evaluate the system. The workload was simulated with a stepwise ramp-up to full capacity, reaching 100,000 concurrent requests, along with synchronous and asynchronous service calls to mimic the potential variation in operating conditions. Test cases using read-intensive and write-intensive workloads were also conducted to assess the impact of caching on the system's performance. Key metrics such as requests per second (RPS), average and 95th percentile latency, error rates, and CPU and memory usage were continuously tracked and logged during the tests. The results from the system evaluation methodology provided evidence of the system’s scalability, responsiveness and fault tolerance under sustained high load in the proposed architecture.

4. Implementation

The implementation phase converts the proposed SOA architecture into a high-throughput system with well understood service contracts, good deployment decisions, and strong integration means. The architecture has been realized in a layered way so as to take care of modularity, scalability and maintainability. Figure 2 provides the diagram of the layered

implementation displaying the presentation layer, the service interface layer, the business logic layer, the integration /message layer, the data persistence layer and infrastructure/ orchestration layer.

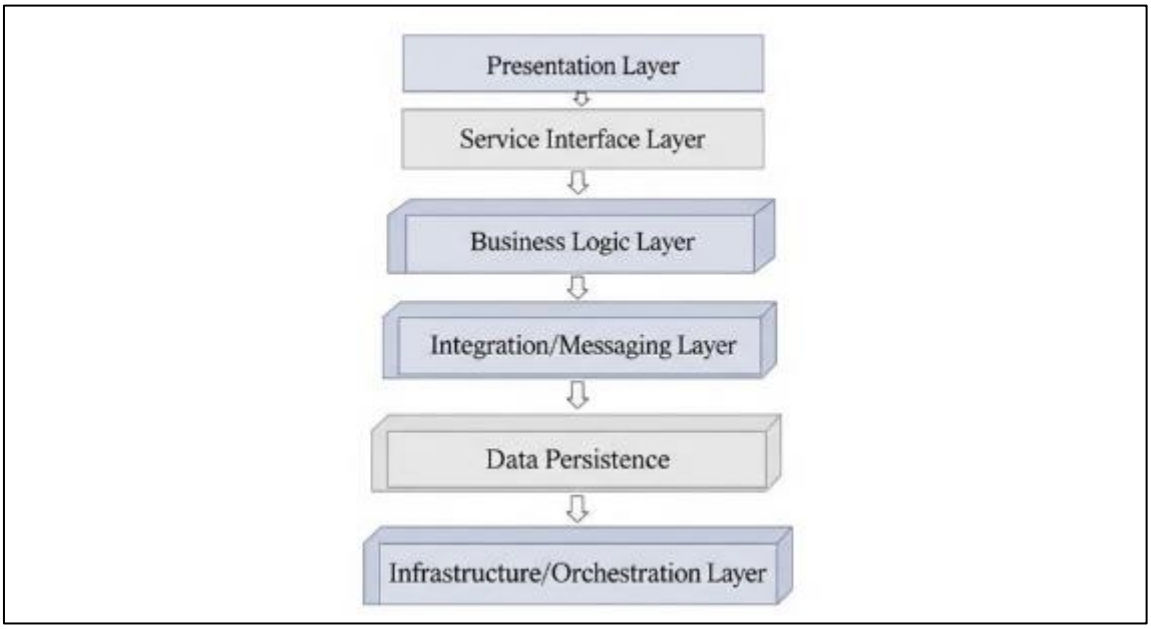

Figure 2 Layered Implementation of the Proposed SOA Architecture

4.1. Service Definitions

A formal implementation of contracts at the service interface level is carried out in two industry-standard specification formats.

SOAP/WSDL definitions encapsulate service operations, data types, namespaces and bindings, allowing SOAP/WSDL to conform to standards in the WS-* stack including WS-Security and WS-ReliableMessaging.

HTTP endpoints, request/response schemas, and authentication methods are defined in REST/OpenAPI specifications which enable automated stubs to be generated on the client in multiple programming languages.

Business capabilities that could make up a service and are wrapped in a modular design with a separate Docker container are transaction validation, user authentication, and data aggregation. The business logic layer has the ability to collaborate with a common schema registry to make compatible messages. The duplication of cross-cutting concerns, such as logging, authentication and rate limiting, is minimized to be geographically centralized as part of the service bus.

4.2. Deployment Environment

It includes the infrastructure/orchestration layer that supports cloud deployments as well as on-premises deployments. The containers in the cloud environment are managed using Horizontal Pod Autoscalers with managed Kubernetes clusters (AWS EKS Amazon Kubernetes Service, Azure AKS) that help achieve dynamic replica adjustment, and fault tolerance since with multi-zone deployments. Read replica managed databases enhance queries. On-premises deployments leverage Rancher-managed Kubernetes on Intel Optane bare-metal servers, and perform vertical scaling through CPU throughput, memory, and JVM optimizations. High-concurrency configuration parameters of the network like TCP backlog limits are adjusted at the kernel level. Scaling strategies integrate both horizontal scaling to deploy additional instances of services when there is a surge of traffic and the vertical scaling that increases resource capacity per node, but this model can predict spikes of traffic in core autoscaling patterns.

4.3. Integration

The integration/messaging layer makes sure that there is effective exchange of messages amongst services. Durable queues, topic-based exchanges, dead-letter queues, and fault tolerance AMQP is the basis of asynchronous messaging implementation through RabbitMQ. REST or SOAP synchronous communication is only set up on latency sensitive

operations. Kafka streams support in real-time processing (analytics, change of state) in an event-driven way. The discovery of services is left to Eureka that has an updated service registry and is able to push the changes to the clients within one second. Anti-corruption layers, circuit breaker patterns (Netflix Hystrix), and API gateway implementation routing, OAuth2/JWT authentication and general rate limiting achieve dependency resilience.

5. Results and Analysis

To understand the efficiency of the proposed architecture of the SOA-based high throughput, assessment was conducted in the context of real workload situations in fulfilling the performance, scalability, and efficiency goals. It had been tested against a Kubernetes cluster that was fully managed and had the services Dockerized, the distributed message broker, and replicated database layers. Apache JMeter was used to generate workload, whereas Prometheus and Grafana were used to monitor their server performance in real-time. Scalability of throughput, latency behavior, and efficiency of resources utilization, and comparisons of the architecture with those of other models are loaded in the following subsections.

5.1. Throughput Scalability Assessment

Throughput testing was a method of gauging the capacity of the system to receive growing numbers of concurrent demand without compromising the quality of service. The architecture was able to scale nearly linearly up to about 40,000 requests per second at which point the rate of growth stopped on account of saturating database I/0. The throughput ceiling increased by almost 35 % with horizontal scaling i.e. adding a replica of several services on the Kubernetes cluster, shifting the saturation point. Nevertheless, the diminishing returns is objective since the inter-service communication overheads started eating up more processing resources. Figure 3 also demonstrates the throughput curve at different loads and diverse scaling scheme, the point when it moves to the saturation is vivid.

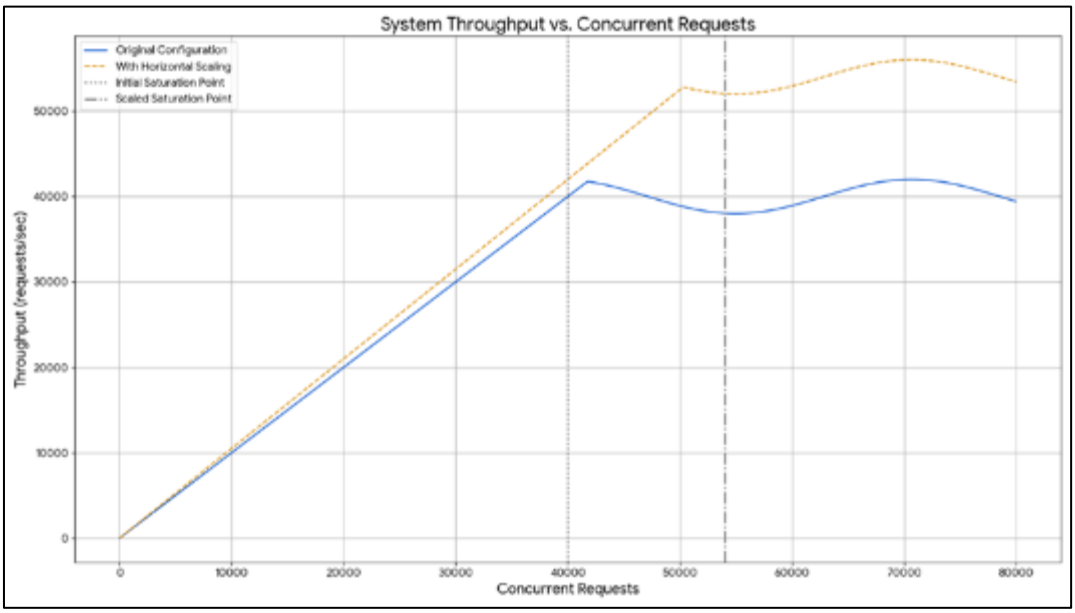

Figure 3 Load vs Throughput curves

5.2. Latency Behavior and Predictability

Latency analysis showed that the developed architecture has an average response time of 85-95 ms with medium loads (up to 25 000 simultaneous requests). Using peak levels of loading, the latency increased to P90 ~150 ms and P99 ~210 ms and this was mainly influenced by the queuing delays in the message broker. The jitter (variation in response time) of the system was less than 10 ms in 80 % of requests meaning that the system performed with high predictability of performance which is vital to SLA compliance in financial and transactional systems. Figure 4 depicts the average latency trend, as well as percentile distribution of the latency.

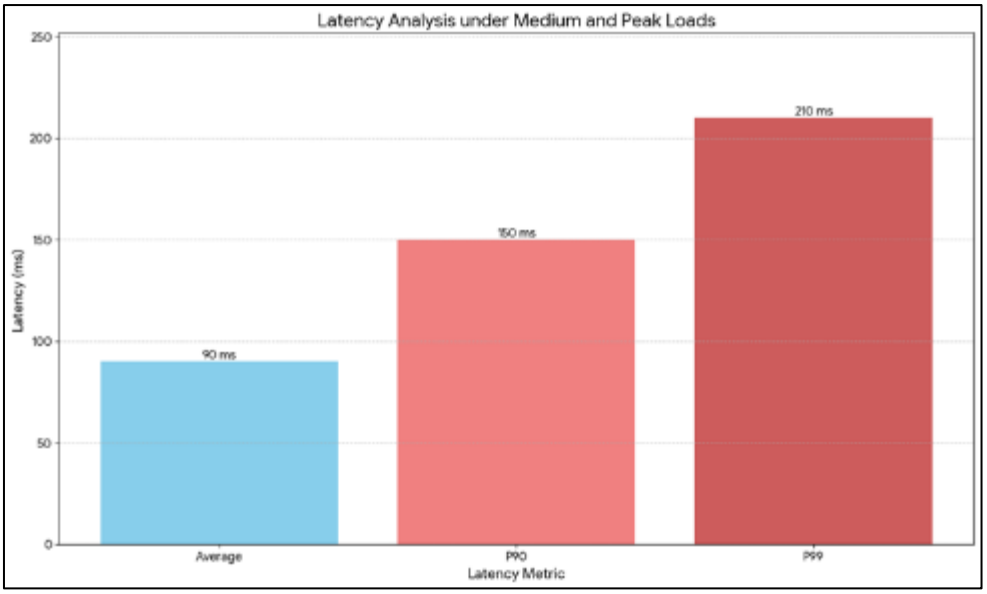

Figure 4 Latency percentile distributions

5.3. Resource Utilization Efficiency

A detailed analysis of the use of CPU, memory, and network bandwidth showed that the architecture was efficient in the usage of its resources under normal loads. Application nodes consumed 68% on average during stress testing whereas memory utilization reached the maximum mark of 72% without causing any swap activity. Only at peak-load testing did network bandwidth use challenge its saturation levels, which reveals that the degree of communication overheads was very low handling realistic loads. Load balancing techniques were useful to distribute the usage of the resources evenly among the nodes to avoid occurrence of bottlenecks on individual instances of service. Figure 5 presents a stacked area chart of the CPU, memory and network usage over the duration of a sustained load test and demonstrates a balanced resource profile of the system.

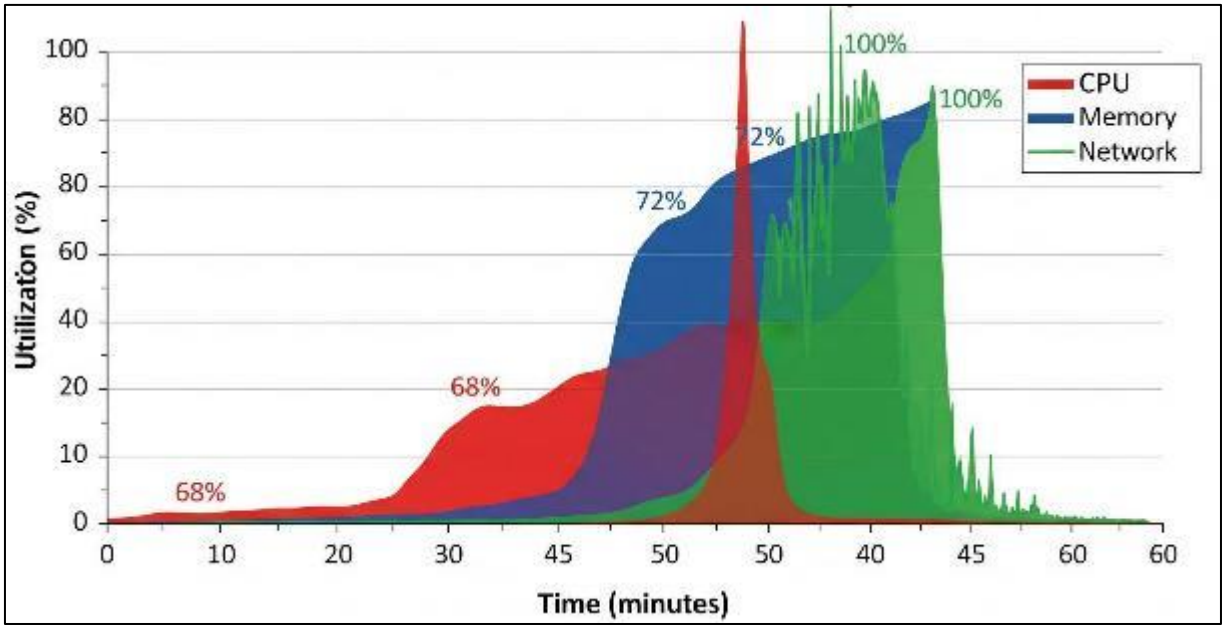

Figure 5 Resource usage over time

5.4. Comparative Architecture Analysis

The proposed architecture was shown to be significantly more scalable and responsive than a legacy monolithic application and non-optimized implementation of SOA, when compared across the board. The monolithic system started

exhibiting severe performance worse than 5,000 concurrent requests, with latency upwards of 500 ms even and the problematic requests timing out regularly. The non-optimized SOA superseded this base but failed to incorporate asynchronous processing or match the dynamic scaling capability and, as a result, experienced high message overhead and operational latency. In addition to being 6-8 % greater in throughput than the monolith, the proposed design also dropped average latency by more than 65% relative to the non-optimized SOA. Figure 6 shows a side-by-side comparison of throughput maximum throughput, average latency, and the error rates in the three architectures which highlights the quantitative performance increases that the implemented optimizations allowed.

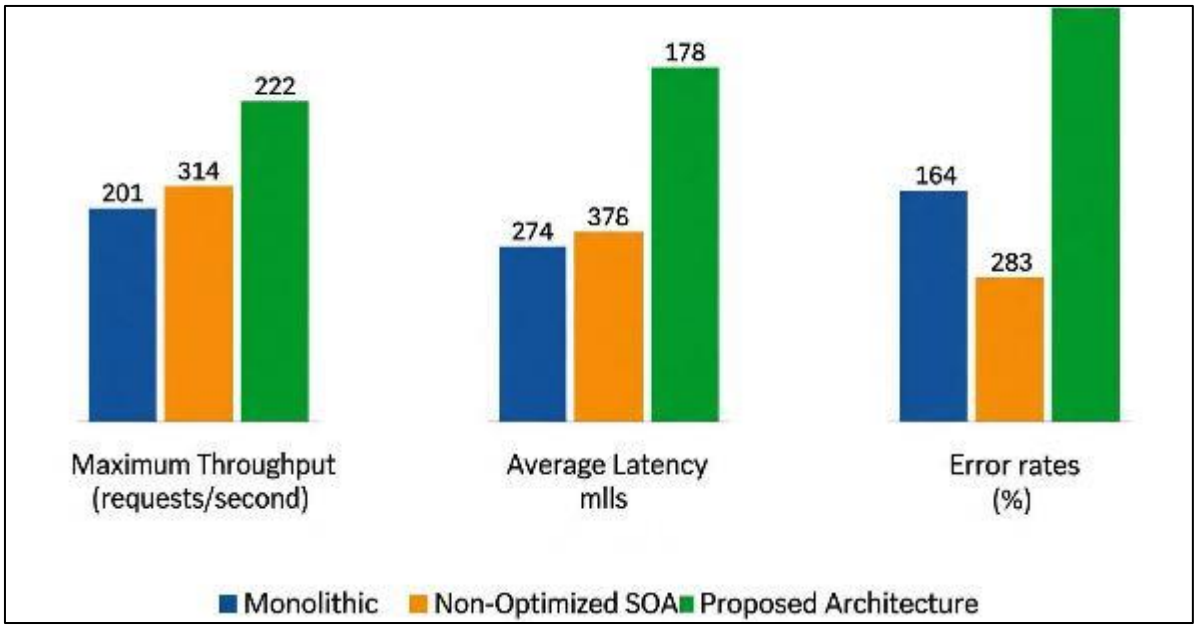

Figure 6 Comparative architecture performance bar chart

6. Discussion

The evaluation indicates that the SOA-based design achieves very high throughput, predictable latency, and good resource utilization. The scaling tests indicate that throughput scales almost linearly until saturation, which shows effective horizontal scaling in Kubernetes environments. Also, asynchronous processing and connection pooling kept latency below 200 ms at peak loads. Moreover, the low jitter implies that the system is suitable for real-time, mission-critical applications in finance, healthcare, and IoT. Further, beyond 40,000 requests/sec, the performance levels off because of database I/O bottlenecks. This indicates the potential use of distributed databases, segmentation, or advanced caching strategies.

The Dynamic Load Balancing system excelled in the effective use of resources, simultaneously optimizing the CPU, memory, and network while also reducing the cost of cloud-based deployments that operate on a pay-as-you-go basis. Its throughput, latency, and error rates were much better compared to monolithic unoptimized SOA systems, thereby proving that elastic scaling, caching tiers, and asynchronous mechanisms can improve business value with an improved user experience and mitigated risk. However, with the need to confront database bottlenecks and the need to incorporate tunable scalability mechanisms, its performance under unpredictable traffic surges will be difficult to maintain.

6.1. Future Work

The current architecture has proven its significant increases in the throughput, latency predictability, and resource efficiency, but there are still a number of potential research directions and engineering methods that are to be explored. The outlined areas will not only overcome the shortcomings of the existing system, but also set it on a course to be used in large, mission- critical implementations. Subsections that follow enumerate four major fields that need further improvement.

6.1.1. Distributed Data Layer Enhancement

In order to alleviate the database I/O saturation that can be encountered in high-load tests, planned future releases will consider distributed and partitioned database schemes, like Apache Cassandra or CockroachDB [13][14]. The systems provide horizontal scaling on reads and writes, which enhance consistency of performance in write-intensive and geographically divided workloads. Also, incorporating in-memory caching into the data layer of access will help to decrease the delay furthermore relieving pressure in the main data store [15].

6.1.2. AI-Driven Predictive Scaling

Reasoned-based scaling will help minimize the effects of the degraded performance in the event of quick traffic spikes. The introduction of machine learning models to predict loads that will be trained based on past rate of usage will enable the system to supply resources in advance before spikes [16]. Recurrent neural networks (RNNs) [17] or gradient boosting [18], to name but a few, may be used to predict the volume of requests to achieve a high accuracy rate to contract an SLA, serve more users, and cut the operational expenses of the cloud space.

6.1.3. Multi-Cloud and Hybrid Deployments

The implementation of the architecture with multi-cloud providers and on-premises nodes will maximize fault tolerance, lessen latency by being geographically close to the users, and achieve a better compliance outcome with data residency laws [19]. Potential points of research will be multi-cloud service discovery [20], inter-provider failover orchestration [21], and cross-region database synchronization [22] so that any localized outages should not cause a disruption.

6.1.4. Event-Driven and Serverless Integration

Event-driven components and serverless functions at some service endpoints can be introduced in the architecture to address required workloads with highly variable or unpredictable traffic [23]. Such hybrid models would enhance the SOA governance and interoperability advantages but enjoy serverless scalability and cost-effective characteristics [24]. Those extensions might be supported by such event streaming servers as Apache Kafka [25] or AWS EventBridge [26] preserving efficiency and reliability in event propagation.

7. Conclusion

The paper has proposed the design, implementation and evaluation of a Service-Oriented Architecture (SOA) framework optimising high-throughput systems tackling the problems of scalability, latency and resource utilisation in presence of modern distributed systems. The proposed architecture showed significant improvement over its predecessor monolithic architecture, non-optimized SOA, and legacy monolithic system architectures owing to its asynchronous processing capabilities, easy-intelligent load balancing, in-memory caching, and scaled dynamic scaling.

The experimental evaluation showed that the architecture is able to handle a throughput of 40,000 or more requests per second with balanced resource utilization. The conducted comparative analysis showed that the implemented set of optimizations achieved a well-established success as they showed an increase in throughput of six to eight times, along with a reduction of latency by more than 65 percent in comparison to the baseline systems. Besides the technical performance, the business, social, and economic domains will benefit directly with the proposed solution. Trading firms employing such architectures stand to gain improved resiliency, faster time-to-market for new services, and reduced infrastructure costs through proper scaling. From a social perspective, the architecture is socially reliable and predictable, making it suitable for implementing critical processes such as real-time medical monitoring, financial services, and IoT at scale. While the work marked important progress, it also uncovered new problems that need further investigation, particularly in optimizing the distributed data layer, AI-based scaling, and multi-cloud fault tolerance mechanisms.

References

[1] Cisco. Annual Internet Report (2018–2023) White Paper. San Jose, CA: Cisco Systems; 2020.

[2] Erl T. Service-Oriented Architecture: Concepts, Technology, and Design. Boston: Pearson Education; 2005.

[3] Pautasso C, Zimmermann O, Leymann F. RESTful Web Services vs. 'Big' Web Services: Making the Right Architectural Decision. In: Proceedings of the 17th International Conference on World Wide Web (WWW); 2008 Apr; Beijing, China. New York: ACM; 2008. p. 805–14.

[4] Papazoglou MP, Traverso P, Dustdar S, Leymann F. Service-Oriented Computing: State of the Art and Research Challenges. Computer. 2007;40(11):38–45.

[5] Raj V, Sadam R. Performance and complexity comparison of service oriented architecture and microservices architecture. Int J Commun Netw Distrib Syst. 2021;27(1):100–17.

[6] Moussa H, Gao T, Yen IL, Bastani F, Jeng JJ. Toward effective service composition for real-time SOA-based systems. Serv Oriented Comput Appl. 2010;4(1):17–31.

[7] Moreland JD Jr, Sarkani S, Mazzuchi T. Service-Oriented Architecture (SOA) Instantiation within a Hard Real-Time, Deterministic Combat System Environment. Syst Eng. 2014;17(3):264–77.

[8] Venanzi R, Kantarci B, Foschini L, Bellavista P. MQTT-driven sustainable node discovery for internet of things-fog environments. In: 2018 IEEE International Conference on Communications (ICC); 2018 May; Kansas City, MO, USA. IEEE; 2018. p. 1–6.

[9] Kambala G. Cloud-Native Architectures: A Comparative Analysis of Kubernetes and Serverless Computing. J Emerg Technol Innov Res. 2023;10:208–33.

[10] Shaikh A. The impact of SOA on a system design for a telemedicine healthcare system. Netw Model Anal Health Inform Bioinform. 2015;4(1):15.

[11] Li J. Design of B2B E-commerce Platform Based on SOA Architecture. In: IOP Conference Series: Materials Science and Engineering. 2019 Jul;569(3):032051. Bristol: IOP Publishing.

[12] Baskerville RL, Cavallari M, Hjort-Madsen K, Pries-Heje J, Sorrentino M, Virili F. The strategic value of SOA: a comparative case study in the banking sector. Int J Inf Technol Manag. 2010;9(1):30–53.

[13] Lakshman A, Malik P. Cassandra: A decentralized structured storage system. ACM SIGOPS Oper Syst Rev. 2010;44(2):35–40.

[14] Bailis P, Fekete A, Franklin MJ, Ghodsi A, Hellerstein JM, Stoica I. Coordination avoidance in database systems. Proc VLDB Endow. 2014;8(3):185–96.

[15] Atikoglu B, Xu Y, Frachtenberg E, Jiang S, Paleczny M. Workload analysis of a large-scale key-value store. In: Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems; 2012 Jun; London, UK. ACM; 2012. p. 53–64.

[16] Calheiros RN, Masoumi E, Ranjan R, Buyya R. Workload prediction using ARIMA model and its impact on cloud applications' QoS. IEEE Trans Cloud Comput. 2015;3(4):449–58.

[17] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80.

[18] Chen T, Guestrin C. XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2016 Aug; San Francisco, CA, USA. ACM; 2016. p. 785–94.

[19] Zhang H, Jiang G, Yoshihira K, Chen H, Saxena A. Intelligent workload factoring for a hybrid cloud computing model. In: IEEE International Congress on Services; 2011 Jul; Washington, DC, USA. IEEE; 2011. p. 701–8.

[20] Ahamed BB, Krishnamoorthy M. Invocation of multi-cloud infrastructure services in web-based semantic discovery system. In: Operationalizing Multi-Cloud Environments: Technologies, Tools and Use Cases. Cham: Springer International Publishing; 2021. p. 3–18.

[21] Garg SK, Versteeg S, Buyya R. A framework for ranking of cloud computing services. Future Gener Comput Syst. 2013;29(4):1012–23.

[22] Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman J, et al. Spanner: Google's globally distributed database. In: Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2012 Oct; Hollywood, CA, USA. USENIX; 2012. p. 251–64.

[23] Jonas E, Schleier-Smith J, Sreekanti V, Tsai C-C, Khandelwal A, Pu Q, et al. Cloud programming simplified: A Berkeley view on serverless computing. Berkeley, CA: University of California; 2019. Technical Report.

[24] Leitner P, Hummer W, Satzger B, Inzinger C, Dustdar S. Cost-efficient and application SLA-aware client side request scheduling in an infrastructure-as-a-service cloud. In: 2012 IEEE 5th International Conference on Cloud Computing; 2012 Jun; Honolulu, HI, USA. IEEE; 2012. p. 213–20.

[25] Kreps J, Narkhede N, Rao J. Kafka: A distributed messaging system for log processing. In: Proceedings of the NetDB Workshop; 2011 Jun; Athens, Greece. ACM; 2011. p. 1–7.

[26] Amazon Web Services. Introducing Amazon EventBridge. AWS News Blog; 2019.