ReTEC

Revista Electrónica de
Tecnología, Educación y Ciencia
ISSN: 2953-5654
http://retec.unsa.edu.ar
Universidad Nacional de Salta

Análisis de la Comunicación Asíncrona en Arquitectura de Microservicios

Nelson Rodríguez, María Murazzo, Hernán Atencio, Marcelo Moreno, Diego Medel

Departamento de Informática, Facultad de Ciencias Exactas Físicas y Naturales
Universidad Nacional de San Juan
nelson@iinfo.unsj.edu.ar , maritemurazzo@gmail.com , hernan.atencio.98@gmail.com
mpmoren@gmail.com , mdiego88@gmail.com

Revista Electrónica de Tecnología, Educación y Ciencia,

Volumen 1, Número 3, pág. 46-54, jun, 2026. ISSN: 2953-5654

Disponible en http://retec.unsa.edu.ar/

Análisis de la Comunicación Asíncrona en Arquitectura de Microservicios

Nelson Rodríguez, María Murazzo, Hernán Atencio, Marcelo Moreno, Diego Medel

Departamento de Informática, Facultad de Ciencias Exactas Físicas y Naturales
Universidad Nacional de San Juan
nelson@iinfo.unsj.edu.ar , maritemurazzo@gmail.com , hernan.atencio.98@gmail.com
mpmoren@gmail.com , mdiego88@gmail.com

Resumen: Las aplicaciones distribuidas han revolucionado las prácticas de desarrollo de software al descentralizar los componentes, facilitar la escalabilidad y permitir la agilidad en el diseño y despliegue de sistemas. Varios problemas como la pérdida de control, la seguridad, la comunicación eficiente entre componentes y la tolerancia a falla aún siguen sin resolverse completamente, lo cual la presentan como un área de investigación desafiante. Además, los entornos distribuidos pueden ocasionar otras problemáticas adicionales, como sincronización, escalabilidad, consistencia, optimización del rendimiento, entre otros. Los microservicios son arquitecturas de software esenciales para aplicaciones distribuidas modernas, pero requieren un enfoque disciplinado en diseño, seguridad, observabilidad y automatización. Un factor sumamente importante es la comunicación, dado que la misma en las aplicaciones monolíticas se realiza entre métodos que forman parte de un solo proceso, es decir, se crea una clase y se llama al método dentro del módulo de destino, todos ejecutando el mismo proceso. Esta comunicación es muy simple pero al mismo tiempo los componentes están altamente acoplados entre sí y son difíciles de separar y escalar de forma independiente. A diferencia de esto, los microservicios presentan variedad de alternativas para comunicarse tanto sincrónica como asincrónicamente, con lo cual la utilización de recursos y los valores de parámetros como performance resultan de interés para el desarrollo de aplicaciones más eficientes y resilientes, especialmente cuando numerosos servicios pequeños colaboran para lograr una actividad empresarial unificada. Debido a que la comunicación asincrónica puede resultar dependiente del bróker o agente de mensajes, el análisis se realiza de forma independiente del mismo. Por lo tanto, el presente trabajo tiene por objetivo analizar las alternativas de comunicación asincrónicas, su implementación, análisis, conclusiones y posibles trabajos futuros.

Palabras clave: Microservices, Distributed Computing, Communication microservices, Synchronous Communication Microservices.

1. Introduction

Los sistemas distribuidos han surgido como herramientas indispensables para abordar las crecientes demandas de potencia de cálculo, escalabilidad y utilización eficiente de los recursos, que resulta imposible de obtener mediante un modelo centralizado. Además recientemente, el rápido crecimiento de las cargas de trabajo de inteligencia artificial, ha impulsado la necesidad de sistemas de computación capaces de procesar conjuntos de datos que superen escalas de petabytes, como los necesarios para entrenar grandes modelos de lenguaje, que implican cientos de miles de millones de parámetros [1].

Los microservicios son un desarrollo que ofrecen muchas ventajas en comparación con las antiguas arquitecturas monolíticas. Por eso, muchas de las grandes empresas tecnológicas han logrado con éxito realizar la transición a microservicios [2].

El tamaño del mercado de microservicios en la nube ha alcanzado los 2.270 millones de dólares en 2025. Se espera que crezca hasta 5.820 millones de dólares en 2030 con una tasa de crecimiento anual compuesta (CAGR) del 20,7% [3].

Este crecimiento está impulsado por empresas que trasladan aplicaciones de aplicaciones monolíticas a arquitecturas modulares nativas de la nube. Alrededor del 62,3% de las organizaciones informaron del uso de microservicios y tecnologías de contenedores para apoyar una codificación más flexible y escalar el desarrollo de aplicaciones y el despliegue de software más rápido en entornos distribuidos [4].

El término microservicios fue acuñado por Peter Rodgers en 2005. La idea principal era dividir los diseños "monolíticos" grandes en múltiples componentes o procesos independientes, lo que hacía el código fuera más granular y manejable [5]. Sin embargo, pasó un tiempo para que el término se difundiera y fue gracias a Martin Fowler en 2014. En su artículo, describe cómo Netflix adoptó una arquitectura de microservicios para mejorar su capacidad de respuesta ante los cambios del mercado [6].

El estudio de aplicaciones distribuidas en arquitectura de microservicios es de gran relevancia debido a su impacto en distintos ámbitos: científico, tecnológico, social y estratégico. Desde una perspectiva científica, este estudio permite profundizar en los principios de diseño modular, comunicación distribuida y gestión eficiente de datos en entornos complejos. Analizar los desafíos técnicos y las mejores prácticas proporciona evidencia empírica que contribuye al conocimiento sistemático sobre patrones de diseño, escalabilidad, tolerancia a fallos y eficiencia en sistemas distribuidos, fortaleciendo el corpus académico en ingeniería de software y tecnologías cloud-native [7].

En el ámbito tecnológico, la investigación es crucial para optimizar el desarrollo de aplicaciones modernas que deben responder a demandas de alta disponibilidad, escalabilidad y resiliencia. La adopción de microservicios permite reducir tiempos de despliegue, facilitar la integración continua y mejorar la eficiencia operativa, lo que impacta directamente en la competitividad de organizaciones y empresas tecnológicas. Identificar estrategias óptimas de comunicación, escalabilidad y resiliencia permitirá implementar soluciones de software más robustas, rápidas y confiables [8].

Desde una perspectiva social, los sistemas basados en microservicios sustentan plataformas que afectan directamente la vida cotidiana de los usuarios, tales como servicios financieros digitales, comercio electrónico, educación virtual y salud en línea. Mejorar la disponibilidad, eficiencia y seguridad de estas aplicaciones contribuye a la calidad del servicio, la confianza de los usuarios y la reducción de interrupciones en actividades críticas, generando un beneficio tangible para la sociedad [9].

Los microservicios presentan algunas limitaciones debido a que los programas distribuidos son más difíciles de programar, las llamadas remotas son lentas, se enfrentan a fallas y se debe realizar el mantenimiento de código en diversos lenguajes y frameworks. Cada servicio requiere pruebas y monitoreo individualizados, lo que garantiza que los sistemas de automatización también deban ser tenidos en cuenta. No es sencillo lograr la coherencia de los datos, porque cada proveedor tiene su propia base de datos y sistema de gestión de transacciones [10]. El mecanismo de comunicación entre servicios es uno de los aspectos más importantes que se deben considerar [11].

2. Comunicación en microservicios

La aplicación basada en microservicios es un sistema distribuido que ejecuta múltiples procesos y servicios, por lo que se pueden utilizar diferentes tecnologías de comunicación. El diseño de la comunicación entre servicios es muy importante al migrar de un software monolítico a una arquitectura de microservicios [12].

En caso de tener que adoptar un método IPC para un microservicio, es importante tener en cuenta si la comunicación entre ellos es sincrónica o asincrónica [13].

Existen diferentes estilos de comunicación:

- Bloqueo sincrónico: en este escenario, un microservicio inicia una llamada a otro microservicio y detiene temporalmente sus operaciones mientras espera la respuesta.
- Sin bloqueo asincrónico: el microservicio que inicia la llamada puede continuar procesando, independientemente de si la llamada se recibe o se completa.
- Requerimiento-Respuesta: un microservicio envía una solicitud a otro microservicio, buscando la ejecución de una tarea y anticipa recibir una respuesta que detalla el resultado.
- Conducido por eventos: los microservicios emiten eventos, que otros microservicios consumen y responden a ellos. Quien genera el evento desconoce qué microservicios, si los hay, están consumiendo los eventos que emite.
- Datos comunes: aunque no se reconocen como un estilo de comunicación, los microservicios colaboran compartiendo datos de una fuente de datos común.

Una arquitectura de microservicio integral a menudo incorpora una combinación de estilos de colaboración, y esta suele ser la práctica estándar. Ciertas interacciones se adaptan naturalmente a un formato de solicitud-respuesta, mientras que otras se alinean más con un enfoque basado en eventos.

El presente trabajo, es continuación de uno publicado sobre comunicación sincrónica [14]. Para el análisis se tiene en cuenta solo el tipo de comunicación asincrónica, pero no el estilo de microservicio.

3. Trabajos relacionados

Existen diversas publicaciones que tratan la temática de comunicación interprocesos en la arquitectura de microservicios. Sin embargo toman diferentes aspectos de los analizados en el presente trabajo.

Un trabajo que aporta a la problemática de la comunicación sincrónica y asincrónica de servicios Web, es el publicado por Ahmed Mujić et al. [15]. Dicho trabajo compara la comunicación sincrónica y asincrónica de los servicios en un proceso generador de notificaciones de negocio. Como conclusiones del trabajo figura la complejidad de la comunicación asincrónica como como la coordinación del equipo, la consistencia eventual, la garantía de entrega de mensajes, cambios en la lógica del negocio y el uso de un sistema de intermediación apropiado.

Un aporte interesante es el trabajo publicado por Lei Zhang et al [16], En dicho artículo, proponen una tecnología de comunicación de eficiente llamada RPCX, que utiliza el modelo de red de E/S sin bloqueo y Protobuf como mecanismo de comunicación subyacente y utiliza

tecnología de proxy dinámico y reglas de configuración de anotación para permitir a desarrolladores llamar a los métodos localmente.

El trabajo publicado por Maria Shehzadi et al [17], realiza un análisis crítico sobre la Comunicación entre Procesos (IPC) y evalúa su impacto sobre la base de diversas funcionalidades no relacionadas con el negocio, tales como eficacia del rendimiento, accesibilidad, adaptabilidad y complejidad. Analiza tanto el enfoque sincrónico como asincrónico. Para las pruebas sincrónicas usa la llamada a procedimiento remoto de Google (gRPC), y para asincrónicas la cola de mensajes Rabbit (RabbitMQ). A diferencia de las publicaciones analizadas, el presente trabajo se enfoca en comparar

4. Metodología

Se utilizaron las siguientes herramientas y frameworks: Java 17, Spring Boot 3.0.3, Feign Client, RestTemplate, Unirest, FlywayDB, Postgres, Lombok, JPA Criteria Queries, Hibernate Metamodel, Prometheus, Grafana y Swagger UI.

No se tiene en cuenta aplicar herramientas de orquestación ni de coreografía, ni tampoco el uso de plataformas cloud específicas, tratando en este caso de evaluar la comunicación sin elementos que puedan afectar los resultados obtenidos. Se utilizó Postman [18] para realizar las pruebas.

Se realizó la implementación de 3 patrones de comunicación asincrónica, los cuales se describen a continuación:

- Comunicación simple entre emisor/receptor
- Comunicación encadenada, donde una petición desencadena en una nueva petición, para luego retornar al emisor inicial.
- Comunicación multicast a través de múltiples unicast.

A su vez, estos tres patrones se trabajaron con Web Client y Project Reactor.

Una vez realizado y testeado el código, se procedió al desarrollo de las pruebas en sí, el cual consiste en una ejecución de 500 repeticiones para cada patrón, en su respectiva tecnología.

Las variables a analizar en este caso fueron las siguientes:

- Suma de las request ejecutadas para validar que el patrón se haya ejecutado correctamente.
- Uso de CPU para cada patrón
- Tiempo de demora de cada petición realizada dentro del patrón correspondiente.
- Tiempo máximo de las request (peor ejecución)
- Promedio realizado entre la suma de tiempo de las request y la cantidad de ejecuciones realizadas

5. Resultados

Los resultados obtenidos luego de realizar las pruebas pertinentes se muestran en los gráficos siguientes:

Primeramente, se realizaron las 500 repeticiones para el patrón de comunicaciones simples, o sea, un emisor y un solo receptor, que se aprecia en la figura 1.

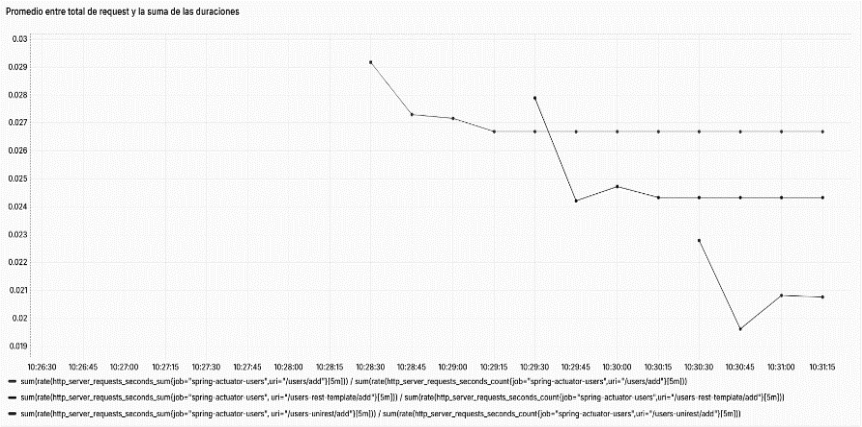

Figura 1. Promedio de Ejecución para flujo simple

El gráfico obtenido muestra el promedio de ejecución para el flujo simple, donde podemos ver que el promedio total de ejecución para el servicio inicial es bastante menor al de su contraparte sincrónica, permitiendo dar pie a la siguiente solicitud de manera más eficiente.

A continuación, la figura 2 muestra los resultados de la ejecución de mircoservicios encadenados.

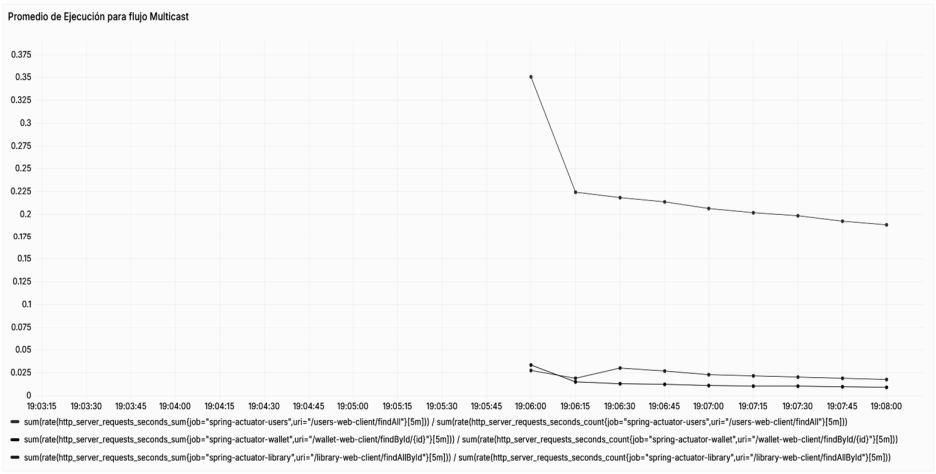

Figura 2. Promedio de ejecución para flujo encadenado

En este caso se encadenan las llamadas para completar el flujo de trabajo y retornar la respuesta, aquí es en donde más se pueden ver las bondades del asincronismo, dado que las llamadas sucesivas se relegan al hilo generado para realizar la petición, aumentando así la disponibilidad del microservicio.

Posteriormente se ejecutaron las pruebas para el flujo multicast y se muestran los resultados para el promedio de ejecución.

Por último, en la figura 3, se aprecia la comunicación multicast.

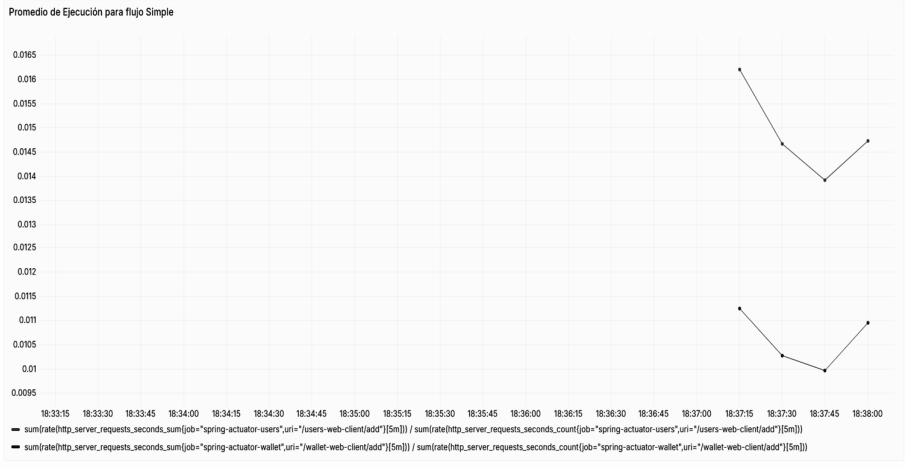

Figura 3. Promedio de ejecución para flujo multicast

Para el flujo multicast, se ejecutaron 5000 requests para el endpoint de wallet/webclient/findAll, 1000 para el endpoint de library/webclient/findAll y 500 para el endpoint de users/webclient/findAll.

Las comunicaciones están implementadas con varios unicast, que solicitan información a varios microservicios para luego retornarla. Para este caso, podemos ver que si bien el asincronismo nos puede llegar a resultar útil como herramienta para paralelizar las peticiones de datos, para poder retornar el conjunto de datos completo, se debe de esperar a que todas las peticiones se ejecuten correctamente, sino, la llamada se retorna con datos incompletos o parciales.

6. Conclusiones y futuros trabajos

Una vez ejecutadas las pruebas, se llegó a las siguientes conclusiones:

Se pudo comprobar que para el caso del flujo encadenado, el tiempo de respuesta promedio se reduce en un 50% respecto de la alternativa sincrónica. Esto dado por la naturaleza asincrónica de los procesos, se puede relegar la finalización de los servicios al hilo correspondiente, permitiendo que la aplicación pueda recibir nuevas peticiones.

A su vez, la implementación asincrónica requiere de un mayor conocimiento comparado con la sincrónica, fundamentalmente sobre conceptos asociados al asincronismo, el manejo de hilos y la ejecución de procesos en segundo plano. Si a esto, particularmente para Java Spring Boot, se le agrega conceptos asociados a la programación reactiva, tanto la cantidad de conocimiento necesario para la correcta utilización de las herramientas, como la curva de aprendizaje de las mismas se vuelve cada vez más elevada.

Se propone como trabajos futuros:

- Finalizar todas las pruebas asincrónicas con un conjunto de valores mayor.
- Analizar el impacto de cada técnica de comunicación en conjunto con los protocolos de seguridad, dado que es una temática que ha sido poco investigada.

- Sobre el estudio presentado, agregar el comportamiento de los lenguajes de programación más utilizados actualmente (Python, Java, Go, NodeJS, por ejemplo).
- Analizar el manejo de transaccionalidad en contextos asíncronos: Se propone expandir la implementación realizada para que esta acapare también los posibles protocolos de manejo de la transaccionalidad asincrónica en arquitecturas basadas en microservicios.
- Evaluar los distintos sistemas de colas de mensajes y agentes de mensajes como como RabbitMQ, Kafka o AWS SQS.
- Analizar cómo influyen en el desempeño los balanceadores de cargar, sin tener en cuenta los nativos del Cloud, herramientas como NGINX o HAProxy distribuyen las solicitudes entrantes entre varias instancias de un servicio para garantizar la confiabilidad y la eficiencia.

Referencias

1. Achiam, J.; Adler, S.; et al. (2023) Gpt-4 technical report. arXiv:2303.08774.
2. Gao, W.; Ren, S.; et al. (2025) Lattice-Based Group Signature with VLR for Anonymous Medical Service Evaluation System. Electronics 2025, 14, 680. https://doi.org/10.3390/electronics14040680
3. Cloud Microservices Market Report 2026.
https://www.thebusinessresearchcompany.com/report/cloud-microservices-global-market-report
4.Cloud Microservices Market Size 2025 – 2034.
https://www.gminsights.com/industry-analysis/cloud-microservices-market.
5. Bennett A.: An Introduction to Microservices. The essential concepts that every developer should know. [Online]. Available: https://medium.com/microservicegeeks/an-introduction-to-microservices-a3a7e2297ee0. (2021).
6. Fowler M.. Microservices Guide. [Online]. Available
felicihttps://martinfowler.com/articles/microservices.html (2014).
7. Velepucha, V., & Flores, P. (2023). Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. IEEE Access.
8. Söylemez, M., Tekinerdogan, B., & Kolukısa Tarhan, A. (2022). Feature-Driven Characterization of Microservice Architectures. Applied Sciences, 12(9), 4424.
9. Castro, L. F. S. de, & Rigo, S. (2023). Relating Edge Computing and Microservices: Systematic Literature Review. arXiv.
10. Chitrak Vimalbhai D., Abhishek P., Utkarsh K.: Microservices Software Architecture: A Review. In: International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429 Volume 9 Issue XI - Available at www.ijraset.com (2021).
11. Almog, D.; Chassidim, H., Shlomo. M.: Testability and Testing of Microservices-complex challenge. In: International Journal of Computers, vol. 6. (2021)
12. Microsoft: Communication in a Microservice Architecture: [Online], Available:
https://learn.microsoft.com/en-us/dotnet/architecture/ microservices/architect-microservice-container-applications/communication-in-microservice-architecture. (2022)
13. K. Bakshi,: Microservices-based software architecture and approaches. In: 2017 IEEE Aerospace Conference, pp. 1–8. doi:10.1109/AERO.2017.7943959 (2017).
14. Atencio H., Rodríguez Nelson, Murazzo María: Evaluación de parámetros asociados a la comunicación síncrona entre microservicios In: 30° Congreso Argentino de Ciencias de la Computación - CACIC 2024.
15. Ahmed Mujić, Nevzudin Buzađija, Samir Lemeš, Denis Čeke: Implementation of an optimized solution to the problems of asynchronous and synchronous communication between web services. In: ELEKTROTEHNIŠKI VESTNIK 92(3): 123-134, 2025.
16. Zhang, L., Pang, K., Xu, J. et al.: High performance microservice communication technology based on modified remote procedure call. In: Sci Rep 13, 12141. https://doi.org/10.1038/s41598-023-39355-4 (2023)

17. Shehzadi M., Riaz N. Chaudhry A.: Inter-Process Communication Amongst Microservices. In: International Journal of Emerging Engineering and Technology (IJEET). ISSN (e): 2958-3764. Vol.: 2, N.: 2, Pages: 1- 8, Year: (2023).
18. Postman: https://www.postman.com