Predicción de Latencia en Microservicios con Modelos de Deep Learning

Ezequiel Lanza[1 [0009−0002−4200−431X]],
Laura Lanzarini[2 [0000−0001−7027−7564]], and
César Estrebou[2 [0000−0001−5926−8827]]

[1] Maestría en Ciencia de Datos,Universidad Austral
emlanza@mail.austral.edu.ar
[2] III-LIDI, Facultad de Informática, UNLP. Centro CICPBA
{laural,cesarest}@lidi.info.unlp.edu.ar

Abstract. La adopción de la tecnología de contenedores está creciendo rápidamente. Los desarrolladores de software encuentran en los contenedores una herramienta que les permite enfocarse en la aplicación en lugar de adaptar diferentes entornos de servidores. Estos son compactos, se ejecutan rápidamente, operan en cualquier entorno, pueden dividirse en módulos pequeños y son autosuficientes. En sistemas a gran escala, gestionar manualmente aplicaciones con cientos o miles de contenedores es complejo, por lo que la orquestación de contenedores es esencial. Actualmente, Kubernetes es el orquestador open source más utilizado, manejando instancias de microservicios (pequeños servicios independientes que se comunican a través de API) y alojándolos en plataformas adecuadas (Pods). Sin embargo, el rendimiento de los microservicios en un clúster de Kubernetes puede degradarse de manera impredecible, generalmente manifestándose en mayores tiempos de respuesta, un indicador clave de confiabilidad. Este artículo presenta la línea de investigación y las tareas que se están desarrollando en el marco de una tesis correspondiente a la carrera de Maestría en Ciencia de Datos de la Universidad Austral (CABA, Bs.As.) cuyo objetivo es predecir la latencia de respuesta de extremo a extremo en una arquitectura de cloud-native (microservicios) sobre un caso de estudio concreto. Actualmente se está trabajando en el desarrollo de un modelo que sea capaz de predecir la latencia futura de un microservicio usando múltiples variables, estudiando especialmente los modelos basados en Transformers que, si bien han demostrado ser útiles para anticipar comportamientos inestables o para la comprensión de texto, aún no se han utilizado para predecir latencias. Se buscará comparar estas arquitecturas con métodos clásicos ampliamente utilizados para esta predicción, utilizando una implementación de referencia de microservicios para el benchmarking de aplicaciones.

1 Motivación

En el ámbito informático, la virtualización se refiere a la asignación de recursos de manera virtual en lugar de física. Esto implica crear recursos virtuales a partir de recursos físicos como CPU, GPU o dispositivos de almacenamiento.

Los primeros sistemas de virtualización surgieron a principios de los 2000 con la aparición de las máquinas virtuales (VMs). Estas se instalan sobre una capa de software llamada hipervisor (como Hyper-V [1] o Vsphere [2]), que controla directamente el acceso a los recursos subyacentes y los asigna a una o más máquinas virtuales. Las VMs permiten ejecutar sistemas operativos completos e independientes, mejorando el manejo del hardware y proporcionando beneficios de seguridad al aislar las VMs unas de otras. Así, un problema en una VM no afecta a las demás que comparten el mismo hardware.

Sin embargo, este enfoque tiene desventajas. Cada VM incluye una imagen de sistema operativo independiente, lo que añade sobrecarga en la memoria y espacio de almacenamiento. Además, la portabilidad de aplicaciones entre diferentes entornos (nubes públicas, nubes privadas y centros de datos tradicionales) es limitada. Estas limitaciones han impulsado la búsqueda de soluciones alternativas, siendo el uso de contenedores la más utilizada actualmente. [3].

Los contenedores son una forma más liviana y ágil de manejar la virtualización permitiendo realizar un aprovisionamiento y una disponibilidad de recursos más rápidamente. En lugar de generar una máquina virtual completa, los contenedores empaquetan todo lo necesario para ejecutar una tarea específica, incluyendo código, dependencias y el sistema operativo, lo que facilita que las aplicaciones se ejecuten en diversos entornos: computadoras de escritorio, infraestructuras de TI tradicionales o la nube. Esto hace que los contenedores faciliten el uso de microservicios.

Los microservicios dividen las aplicaciones más grandes en procesos pequeños, brindando una mayor flexibilidad a los usuarios y separando dichos procesos de manera segura. Ejecutar microservicios en una VM implicaría o bien crear una VM separada para cada microservicio, lo que es una asignación de recursos ineficiente, o bien ejecutar múltiples servicios en la misma VM perdiendo de esta forma los beneficios del aislamiento.

En base a lo antes dicho puede afirmarse que el uso de microservicios es un enfoque arquitectónico y organizativo para el desarrollo de aplicaciones, en el que el software se compone de pequeños servicios independientes que se comunican a través de APIs bien definidas haciendo que cada servicio pertenezca a equipos pequeños e independientes. Esto ha provocado que los servicios web a gran escala (por ejemplo, Netflix, Microsoft Bing, Uber, Spotify, etc.) hayan adoptando principios y patrones de diseño nativos de la nube, como microservicios y contenedores, para aprovechar mejor las ventajas del modelo de entrega, que incluye una mayor agilidad en el despliegue de software, escalabilidad automatizada y portabilidad [4].

Dada su dimensión, estas soluciones son implementadas en grandes centros de datos, por lo cual, si la latencia no es medida, el rendimiento de una aplicación puede verse afectado, interrumpiendo los sistemas de producción generando enormes incidentes visibles para el usuario [5].

En base a lo dicho anteriormente, las investigaciones descriptas en este artículo focalizan en la construcción de un modelo para predecir la latencia extremo a ex-

tremo de una aplicación basada en microservicios capaz de adaptarse a diferentes escenarios.

2 Trabajos relacionados

Pueden encontrarse en la bibliografía varias soluciones a la predicción de la latencia de los microservicios en una plataforma de cloud-native. En [6] se utilizó un sistema para el escalado de recursos llamado RScale, el cual emplea un modelo de rendimiento basado en el aprendizaje automático probabilístico, que puede adaptarse rápidamente a la dinámica cambiante de un entorno de microservicios. En [7] se analizaron y compararon enfoques basados en datos y una variedad de métricas de recursos para predecir la latencia de respuesta de extremo a extremo de un flujo de trabajo de microservicios en contenedores que se ejecuta en Kubernetes, una plataforma con capacidad para administrar cargas de trabajo y servicios en la nube. En [8] se reconoce la necesidad de hacer frente a los requisitos de latencia de los servicios en la nube y se propone eliminar el impacto negativo en el rendimiento de cierta clase representativa de tareas. Para ello, se trabaja con una taxonomía de dichas tareas basada en su activación y en la viabilidad de su control. También los autores proponen el controlador de tareas en segundo plano, BTC (Background Tasks Controller), un enfoque totalmente distribuido para eliminar el impacto negativo de las tareas interactivas en los microservicios en la nube.

En [9] el foco está puesto en las aplicaciones sensibles a la latencia con estrictas limitaciones de retardo entre los distintos componentes ya que plantean retos adicionales en las plataformas. En este artículo se analiza Amazon Web Services (AWS), una de las plataformas más utilizadas actualmente, en busca de las características de retardo de los componentes y servicios clave que afectan al rendimiento general de las aplicaciones sensibles a la latencia. También se define una metodología de medición detallada para plataformas CaaS (Container as a Service).

Realizar un estudio sobre la latencia de los servicios será de utilidad para otorgar visibilidad en términos de la carga de trabajo y cómo esta afecta la performance con vistas al usuario, en lugar de sólo hacer foco en el hardware de la aplicación. Por lo tanto, los resultados de las investigaciones realizadas en el marco de este estudio ayudarán a plataformas como Kubernetes para que la previsión y provisión de recursos sea realizada eficientemente.

La predicción de la latencia últimamente se ha realizado mediante la utilización de redes neuronales del estilo convolucional (CNN) o recurrente (RNN), dado que estos presentaron mejores resultados. Curiosamente, este mismo camino fue transitado por otros problemas, donde el avance en la predicción se dio tras transitar el camino desde métodos tradicionales analíticos a redes neuronales y finalmente con Transformers, como fue el caso del lenguaje natural. Por lo tanto, en el presente estudio se propone realizar el mismo aporte, analizando cómo las arquitecturas de Transformers pueden ser de utilidad para la predicción de las latencias de microservicios. Esto también será un aporte a la gran comunidad

de cloud-native, ya que la principal razón por el crecimiento exponencial en la adopción de esta metodología es debido a la gran comunidad que colabora en encontrar soluciones creativas y de código abierto ante los problemas que se van sucediendo.

3 Solución propuesta

En este artículo se analizan distintos modelos de aprendizaje profundo para predecir la latencia de un microservicio de front end dentro de una arquitectura de cloud-native. Este ha sido elegido por tener impacto directo en las vistas del usuario. Esto no constituye una limitación a los alcances de este trabajo pudiendo seleccionarse cualquier otro microservicio, ya que el objetivo de la solución será demostrar cómo esta se comporta ante determinados cambios.

Para operar con el modelo se trabajó con una arquitectura de referencia ampliamente utilizada por la comunidad de código abierto "Online boutique" presentada inicialmente por Google [10]. Online Boutique consta de una aplicación de microservicios de 11 niveles. Es una aplicación de comercio electrónico basada en la web donde los usuarios pueden buscar artículos, agregarlos al carrito y comprarlos. Google usa esta aplicación para demostrar el uso de tecnologías como Kubernetes/GKE, Istio, Stackdriver y gRPC.

En lo que se refiere a la generación de datos se utilizó el simulador de interacciones Apache Jmeter con el objetivo de obtener diferentes comportamientos.

Del dataset inicial, se utilizó la latencia P95, ya que este método tiene como objetivo ofrecer el mejor equilibrio entre escalabilidad y volatilidad. Este es un método muy utilizado por los proveedores de internet y se ha convertido en el estándar entre ellos. El umbral P95 indica que el 5% de la duración de las transacciones es mayor que el umbral. Por ejemplo, si el umbral P95 es de 50 milisegundos, entonces el 5% de las transacciones superaron ese umbral y tardaron más de 50 milisegundos.

Una vez obtenido el dataset, se realizaron tareas de preprocesamiento y adecuación de los datos para generación y prueba de los distintos modelos deep learning con capacidad de operar sobre datos temporales. A continuación se describen las tareas más relevantes con mayor detalle.

3.1 Preparación del dataset

Para obtener un dataset representativo y variado se simularon, con la utilización del software Apache Jmeter, múltiples interacciones por 2 horas.

El resultado del proceso de simulación fue registrado utilizando una frecuencia de 5 segundos dando como resultado un dataset de 2305 muestras. Cada muestra quedó conformada por 4 tipo de valores: tres de ellos corresponden a valores de latencias, y uno al througput que corresponden a cada microservicio en cada muestra. El "througput" se mide en términos de solicitudes por segundo (RPS) y se puede encontrar en los resultados de la ejecución de una prueba de carga mientras que la latencia se refiere al tiempo que tarda una solicitud en ser

procesada. Dentro de cada muestra, la latencia se presenta en 3 variantes: p50, p95 y p99, correspondientes a los percentiles 50, 95 y 99 respectivamente.

Con esta representación del proceso, en donde cada muestra integra información de latencia y cantidad de solicitudes, se buscó capturar los diferentes aspectos de la distribución de los tiempos de respuesta, desde los valores típicos hasta los valores excepcionales.

Se tomo como variable Target a predecir, el valor de la latencia en p95 del microservicio de "Front End". Se eligió dicho microservicio ya que este será el encargado de interactuar con el usuario y por lo tanto es donde se percibirá algún tipo de retraso en la respuesta de la aplicación.

En las pruebas realizadas, se usaron modelos de Markov [11] para identificar alta varianza en series temporales, mejorando el dataset inicial. Estos modelos, comunes en predicciones financieras, ayudan a entender comportamientos volátiles y a realizar análisis más precisos. Dado que los modelos de Markov identificaron dos regímenes en la serie, se agregó esta información al dataset como embeddings para mejorar la capacidad predictiva del modelo.

3.2 Arquitecturas utilizadas

Dado que el entrenamiento de modelos de deep learning para datos temporales es un proceso computacionalmente costoso, se realizó una primera medición de desempeño de las técnicas habituales. En este caso particular, se consideró el uso de una estrategia estadística y tres modelos de deep learning. En todos los casos se utilizaron secuencias de entrada y de salida de longitud 50, es decir que, cada modelo será capaz de predecir 50 valores de latencia luego de recibir como entrada los 50 previos. Este valor fue fijado heurísticamente y se lo considera aceptable para medir el desempeño de modelos en este caso de estudio. La figura 1 ejemplifica lo antes dicho. Aplicando esta representación a una parte del dataset generado por simulación se obtuvieron 401 secuencias o ventanas.

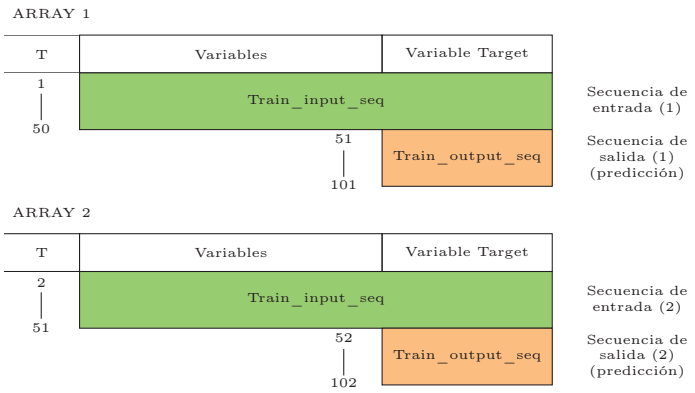

Fig. 1: Estructura de las muestras para entrenamiento y predicción.

La performance de cada modelo fue medida utilizando el Error Cuadrático Medio (MSE).

En las siguientes secciones se describen en detalle los modelos analizados y se presentan los resultados obtenidos.

Modelos Autoregresivos

Se evaluó el desempeño de un modelo VAR (Vectores Autorregresivos) debido a su naturaleza multivariable. En este estudio, cada ventana de datos se analizó de forma independiente para determinar la capacidad del modelo VAR en este tipo de escenarios.

Los modelos VAR requieren que los datos cumplan con condiciones de estacionariedad para su aplicación. Por lo tanto, se consideraron únicamente las secuencias que satisfacían dicha condición, utilizando la prueba de Dickey-Fuller aumentada. Como resultado, el dataset de entrenamiento quedó conformado por 310 ventanas estacionarias.

La aplicación del modelo VAR a este dataset arrojó un Error Cuadrático Medio (MSE) de **1,7**. A continuación, se presentan gráficas comparativas de algunas series obtenidas, contrastando las predicciones con los datos originales.

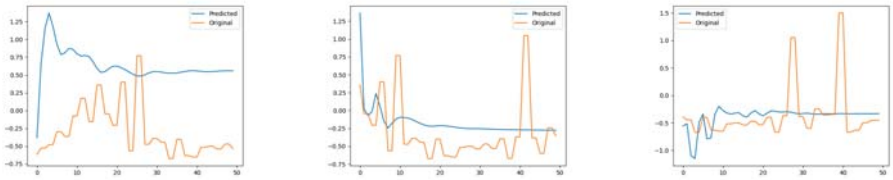

Fig. 2: Muestras de predicción para el modelo VAR.

Dado que los resultados obtenidos no son satisfactorios, tanto en términos de la forma de la curva predicha como de los valores del Error Cuadrático Medio (MSE), se decidió descartar este modelo como alternativa. Los otros métodos evaluados generan resultados más precisos y confiables. Aunque reconocemos el potencial de este enfoque, su correcta implementación y configuración requieren una inversión de tiempo adicional que, en el contexto actual del estudio, no se justifica dada la superioridad de los métodos alternativos.

Red LSTM

Las redes LSTM (Long Short-Term Memory) son reconocidas por su capacidad para aprender relaciones complejas en datos secuenciales. En esta sección, se evalúa la capacidad de este modelo para predecir el valor de latencia.

Tras experimentar con diversas arquitecturas, la configuración seleccionada contó con dos capas LSTM de 128 y 64 unidades respectivamente y una capa densa para la salida. Se aplicó un dropout del 30% luego de las capas LST. Se

utilizó MSE como función de pérdida, optimizador Adam y métricas de MSE, con una parada temprana de 2 épocas de paciencia.

El valor de MSE obtenido fue de **0.534**, significativamente inferior al logrado con el modelo VAR.

A continuación se muestran los gráficos de algunas de las series obtenidas (Predicción vs Original) donde se puede observar que el modelo es capaz de identificar de manera aceptable el comportamiento de la serie haciéndolo una opción valida para su estudio .

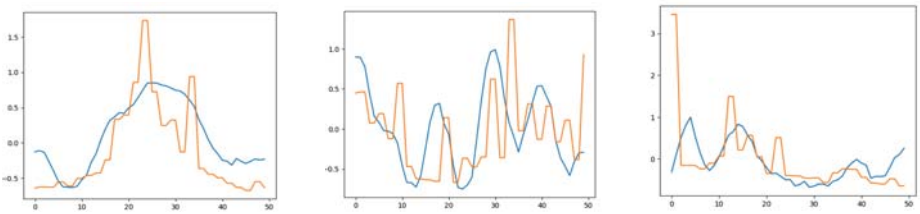

Fig. 3: Muestras de predicción para modelos LSTM.

Red Neuronal Convolucional (CNN)

El modelo CNN se diseñó con capas de convolución de 32 y 64 filtros, seguido de capas de pooling, una capa de aplanamiento, y una capa densa para predicción, compilado con la función de pérdida MSE y optimizador Adam. Además, se implementó una parada temprana para monitorear MSE con paciencia de 2 épocas.

Luego de realizar las pruebas, se obtiene un MSE acumulado de **0,67**, si bien el resultado fue peor comparado con el LSTM, es considerablemente mejor que el autoregresivo.

A continuación se muestran las gráficas de algunas de las series obtenidas (Predicción vs Original) donde se puede observar que, al igual que la LSTM, el modelo es capaz de identificar de manera aceptable el comportamiento de la serie haciéndolo también una opción valida para su estudio.

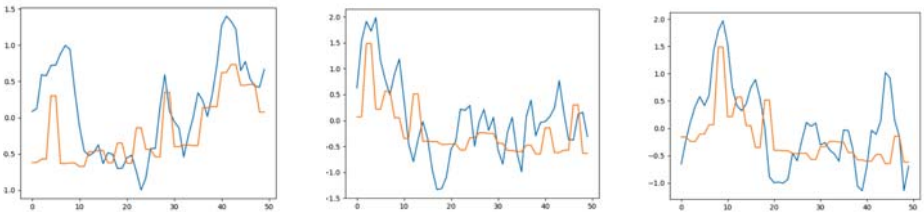

Fig. 4: Muestras de predicción para modelos CNN.

Transformer Informer

Los Transformers han demostrado ser una solución destacada debido a su capacidad para capturar relaciones complejas en datos secuenciales aún con distancias temporales largas. Su mecanismo de multi atención les permite capturar relaciones complejas entre diferentes variables en una serie temporal multivariante. Esto los convierte en una herramienta eficaz para la predicción a largo plazo. Actualmente existen variantes del Transformer, conocidas como X-formers, que abordan desafíos específicos como la eficiencia, la generalización y la adaptación a aplicaciones particulares.

En este estudio se exploraron los beneficios de dos variantes de Transformers: Informer y Spatiotemporal. El Informer modifica la arquitectura estándar al ajustar la representación de entrada con vectores de mayor dimensión y al introducir una codificación posicional jerárquica que mejora la captura de dependencias temporales. La arquitectura de encoder-decoder del Informer también optimiza el procesamiento de secuencias largas y mejora la predicción mediante la destilación de atención. Spatiotemporal, por su parte, extiende el enfoque del Informer al integrar una representación espacial dinámica utilizando una red de grafos dinámica, eliminando la necesidad de definir gráficos estáticos previamente. Este método proyecta las variables en un espacio de mayor dimensión y entrena los embeddings para representar mejor las relaciones espaciales y temporales, mejorando así la precisión de las predicciones. Ambos enfoques demuestran cómo adaptar los Transformers para abordar eficazmente los desafíos específicos de las series temporales.

En este trabajo se utilizó la implementación de Informer [12] disponible en (https://github.com/zhouhaoyi/Informer2020) dado que facilita la configuración de todos los hiperparámentos que pueden influir en el funcionamiento del modelo.

Se realizaron varias ejecuciones hasta encontrar los hyperparametros que mejor resultado arrojaron, ajustando parámetros clave como el tamaño de entrada y salida al encoder/decoder (7 y 1, respectivamente), la dimensión del modelo (512), utilizando una atención ProbSparse con un factor de 5 para manejar secuencias largas. Además, se incorporó una codificación de características de tiempo y técnicas de destilación en el encoder para mejorar la eficiencia y precisión del modelo.

Luego de realizar las pruebas, se obtuvo un MSE de **0,58**, similar al obtenido con LSTM haciendo interesante el estudio posterior para su comparación.

A continuación se muestran las gráficas de algunas de las series obtenidas (Predicción vs Original) donde se puede observar que, al igual que la LSTM, el modelo es capaz de identificar de manera aceptable el comportamiento de la serie haciéndolo también una opción válida para su estudio.

En resumen, observando los valores de MSE arrojados por cada uno delos modelos descriptos previamente, se concluye que el modelo VAR posee un desempeño muy inferior al de los otros tres. La tabla 1 resume lo antes dicho.

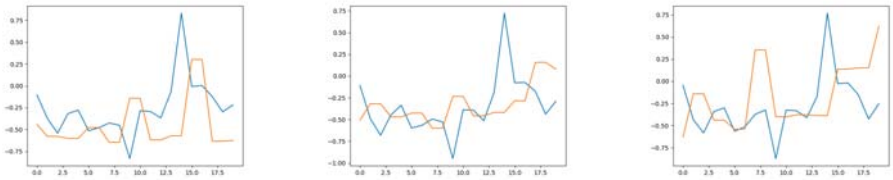

Fig. 5: Muestras de predicción para modelos Informer.

Mean Squared Error (MSE)					
Ventana		Modelos			
Predicción	**Entrenamiento**	**VAR**	**LSTM**	**Informer**	**CNN**
50	50	1,7	0.534	0.58	0.67

Table 1: Resultados obtenidos para la selección de modelos.

4 Experimentos realizados

El desempeño de los tres modelos seleccionados: LSTM, CNN e Informer se midió utilizando como métrica MSE para distintas longitudes de ventanas tanto en la entrada como en la salida. Con esto se busca analizar la capacidad de cada uno para procesar secuencias con distancias temporales largas. Las longitudes medidas fueron 20, 40, 100, 150 y 200.

Los resultados obtenidos, mostrados en la a Tabla 2, permiten afirmar que los modelos LSTM (Long Short-Term Memory) son más eficaces cuando se utilizan ventanas de entrada y predicción cortas. Esto se debe a que los LSTM son capaces de captar patrones temporales a corto plazo de manera eficiente, aprovechando su capacidad para retener información en secuencias de datos limitadas. La naturaleza de los transformers, en contraste, requiere una mayor cantidad de datos para comprender plenamente el comportamiento de las series temporales, ya que dependen de la auto-atención para captar relaciones complejas a largo plazo.

Por otro lado, el modelo Informer demostró ser superior en términos de error cuadrático medio (MSE) cuando se dispuso de ventanas de entrada extensas y se realizaron predicciones a largo plazo. Específicamente, en escenarios con predicciones de hasta 200 muestras, el Informer superó al LSTM en precisión. Esto subraya la ventaja del Informer (Transformer) en manejar secuencias más largas y en capturar patrones de largo plazo que son más difíciles de predecir con modelos recurrentes como LSTM.

Mean Squared Error (MSE)				
Ventana		Modelos		
Predicción	Entrenamiento	LSTM	Informer	CNN
20	20	0.6153	**0.5762**	0.8035
	80	**0.3984**	0.4492	0.5787
	150	**0.5409**	0.7018	0.6222
	250	0.5805	**0.5638**	0.6361
	300	**0.5239**	0.5305	0.8433
60	20	0.5605	**0.4491**	0.6717
	80	**0.3636**	0.4931	0.6342
	150	**0.4271**	0.6009	0.7344
	250	**0.4913**	0.5916	0.6596
	300	**0.408**	0.4677	0.7126
120	20	**0.5006**	0.5315	0.5236
	80	**0.3887**	0.6136	0.6152
	150	**0.4487**	0.5379	0.7115
	250	0.5028	**0.4220**	0.7614
	300	0.4544	**0.3834**	0.7714
200	20	**0.4758**	0.6304	0.5720
	80	**0.4104**	0.4964	0.5842
	150	0.4287	**0.428**	0.7241
	250	0.4115	**0.3832**	0.6860
	300	0.4458	**0.3645**	0.6419

Table 2: Resultados obtenidos

5 Conclusiones

En este artículo se analizaron distintos modelos con capacidad para operar con series temporales con el objetivo de predecirla latencia de microservicios.

Los resultados obtenidos dejan ver que ninguno de los modelos analizados, por si sólo, obtuvo el menor valor de MSE en todas las pruebas realizadas. En base a esto y en el contexto del caso de estudio planteado en este trabajo, la solución más adecuada parece ser el uso de un "selector de modelos" que actuaría como un sistema de identificación eligiendo el modelo más adecuado según el contexto y el momento de la predicción.

Por ejemplo, al inicio del sistema, cuando no hay información histórica disponible, realizar una predicción acertada puede ser extremadamente difícil. En esta fase, se podría considerar el uso de modelos autoregresivos, que son adecuados para situaciones con datos limitados. Conforme el sistema acumula datos históricos, se puede comenzar a utilizar un modelo LSTM para las primeras predicciones, ya que este modelo es eficaz cuando se dispone de poca información y la ventana de predicción es corta. Finalmente, a medida que se acumula una cantidad significativa de datos, un modelo basado en transformers se convierte en la opción indicada, debido a su capacidad para manejar grandes cantidades de información y realizar predicciones a largo plazo con mayor precisión.

En resumen, el selector de modelos optimiza la predicción al adaptarse dinámicamente a las condiciones de los datos disponibles, garantizando así que el modelo seleccionado sea el más adecuado para las características temporales y la extensión del horizonte de predicción en cada momento dado. Esta estrategia modular no solo maximiza la precisión de las predicciones, sino que también aporta flexibilidad y adaptabilidad al sistema de previsión, permitiendo una transición fluida entre diferentes modelos conforme evoluciona la cantidad de datos.

References

1. J. Kappel, A. Velte, and T. Velte, *Microsoft Virtualization with Hyper-V: Manage Your Datacenter with Hyper-V, Virtual PC, Virtual Server, and Application Virtualization*. Network professional's library, McGraw Hill LLC, 2009.
2. F. Guthrie, S. Lowe, and M. Saidel-Keesing, *VMware VSphere Design*. USA: SYBEX Inc., 1st ed., 2011.
3. A. Randal, "The ideal versus the real: Revisiting the history of virtual machines and containers," *ACM Comput. Surv.*, vol. 53, feb 2020.
4. M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, "Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures," *Serv. Oriented Comput. Appl.*, vol. 11, p. 233–247, jun 2017.
5. Y. Zhang, D. Meisner, J. Mars, and L. Tang, "Treadmill: Attributing the source of tail latency through precise load testing and statistical inference," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 456–468, 2016.
6. P. Kang and P. Lama, "Robust resource scaling of containerized microservices with probabilistic machine learning," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pp. 122–131, 2020.
7. H. Mohamed and O. El-Gayar, "End-to-end latency prediction of microservices workflow on kubernetes: A comparative evaluation of machine learning models and resource metrics," in *54th Hawaii International Conference on System Sciences*, pp. 1717–1726, 2021.
8. D. Fireman, *Improving tail latency of interactive cloud microservices through management of background tasks*. PhD thesis, Universidade Federal de Campina Grande. Brasil, July 2021.
9. I. Pelle, J. Czentye, J. Dóka, and B. Sonkoly, "Towards latency sensitive cloud native applications: A performance study on aws," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 272–280, 2019.
10. Google, "Online Boutique." https://github.com/GoogleCloudPlatform/microservices-demo, 2022. Accedido el 7 de diciembre de 2022.
11. M. Awiszus and B. Rosenhahn, "Markov chain neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2180–2187, 2018.
12. H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," Mar. 2021. arXiv:2012.07436 [cs].