ISBN 978 -987-633-574-4

Análisis de ejecución múltiple de Funciones Serverless en AWS

Nelson Rodríguez, Hernán Atencio, Martín Gómez, Lorena Parra, Maria Murazzo

Departamento de Informática, Facultad de Ciencias Exactas Físicas y Naturales,
Uniuversidad Nacional de San Juan, San Juan, Argentina
nelson@iinfo.unsj.edu.ar, hernan.atencio.98@gmail.com, martinsj0811@gmail.com,
lorenaparra152@yahoo.com.ar,maritemurazzo@g,mail.com

Resumen. Serverless Computing es una reciente arquitectura para Cloud Computing que presenta ventajas considerables para los usuarios. Sin embargo debido a recientemente aparición, muchas de sus limitaciones o desventajas no están totalmente resueltas. Numerosos desafíos presenta esta arquitectura, que son analizados y estudiados tanto por la academia como por las empresas proveedoras de Cloud. Las plataformas Serverless permiten ejecutar funciones individuales que pueden ser administradas y ejecutadas separadamente. A diferencia de las aplicaciones tradicionales de ejecución prolongada en plataformas dedicadas, virtualizadas o basadas en contenedores, las aplicaciones serverless están diseñadas para crear instancias cuando se les llama, ejecutar una sola función y cerrarse cuando finalizan. La ejecución eficiente y de alta performance es uno de los desafíos a resolver. Aunque las funciones se ejecutan sin estado y escalan bajo demanda, la ejecución múltiple de funciones requiere que varias dificultades sean resueltas, entre ellas la latencia que presentan antes de ser ejecutadas y que impactan en la eficiencia. En el presente trabajo se realiza una serie de pruebas y análisis de los resultados, que permiten emitir conclusiones sobre el impacto que causa la ejecución de múltiples funciones.

Keywords: Serverless Computing, FaaS, Function-as-a-service, Cloud Computing

1 Introducción

La arquitectura serverless es un modelo de computación en el Cloud impulsado por eventos en el que los recursos informáticos se proporcionan como servicios escalables. En el modelo tradicional se cobraba un costo fijo y recurrente por los recursos informáticos del servidor, independientemente de la cantidad de trabajo informático realizado por el servidor. Sin embargo, la implementación de la computación Serverless ha superado esta deficiencia, ya que permite a los clientes pagar solo por el uso del servicio y no se cobran costos ocultos asociados con el tiempo de inactividad.

En este paradigma emergente, las aplicaciones de software se descomponen en múltiples funciones independientes sin estado [1] [2]. Las funciones solo se ejecutan

en respuesta a acciones desencadenantes (como interacciones de usuario, eventos de mensajería o cambios en la base de datos), y se pueden escalar de forma independiente y pueden ser efímeras (pueden durar una invocación) y están completamente administrados por el proveedor de Cloud.

Los principales proveedores de nube han propuesto diferentes plataformas informáticas sin servidor como AWS Lambda, Microsoft Azure Functions y Google Functions. Dichas plataformas facilitan y permiten que los desarrolladores se centren más en la lógica de negocios, sin la sobrecarga de escalar y aprovisionar la infraestructura, ya que el programa se ejecuta técnicamente en servidores externos con el apoyo de proveedores de servicios en el Cloud [8].

No existen muchas definiciones de Serverless. Una de estas fue publicada por Castro p., Ishakian v., Muthusamy v. y Slominski [6], la cual ofrece una descripción de la características: "La informática serverless se puede definir por su nombre, que es pensar (o preocuparse) menos por los servidores. Los desarrolladores no necesitan preocuparse por los detalles de bajo nivel de administración y escalado de servidores, y solo pagan cuando procesan solicitudes o eventos". Luego la define como: "La informática serverless es una plataforma que oculta el uso del servidor a los desarrolladores y ejecuta código que escala bajo demanda automáticamente y facturado solo por el tiempo que se ejecuta el código".

Debido a que la implementación se realiza mediante la plataforma de funciones, la computación serverless, también es denominada función como servicio (FaaS). En este enfoque, casi todas las preocupaciones operativas son abstraídas lejos de los desarrolladores. Los cuales en principio simplemente escriben código e implementan sus funciones en una plataforma sin servidor. La plataforma se encarga de la ejecución de la función, el almacenamiento y la infraestructura de contenedor, redes y tolerancia a fallas. Adicionalmente, también se encarga de escalar las funciones según la demanda real.

En la mayoría de los casos, se pueden escribir funciones en el lenguaje que el programador considere más adecuado (Node.js, Python, Go, Java y más) y utilizar herramientas de contenedor y serverlessr, como AWS SAM o la CLI de Docker, para compilar, probar e implementar las funciones.

El tamaño del mercado global de Serverless Computing se valoró en $ 3.1 millones en 2017 y se proyecta que alcance casi $ 22 millones para 2025, según un informe de Investigación y Mercados que pronostica hacia dónde se dirige la arquitectura Serverless, para 2025 [16]

Un modelo basado en funciones es particularmente adecuado para ráfagas, uso de CPU intensivo, cargas de trabajo granulares. Actualmente, los casos de uso de FaaS varían ampliamente, incluido el procesamiento de datos, el procesamiento de flujo, la computación de borde (IoT) y la computación científica [3]. Con la continua experimentación generalizada en torno a FaaS, es probable que otros casos de uso surjan en un futuro cercano.

AWS Lambda, popularizó en 2014 el concepto de informática serverless, que permiten a los desarrolladores escribir un fragmento de código que realiza una determinada tarea, ejecutarlo en el Cloud y no preocuparse por administrar la infraestructura subyacente.

AWS Lambda de Amazon [1] fue la primera plataforma serverless y definió varias dimensiones clave que incluyen costo, modelo de programación, implementación,

límites de recursos, seguridad y supervisión. Los lenguajes soportados incluyen Node.js, Java, Python y C#. y se pueden usar herramientas de contenedor y sserverless como AWS SAM o la CLI de Docker, para compilar, probar e implementar las funciones.

En sus orígenes, en Lambda cada fragmento de código suele realizar una única tarea. Por eso en 2017, AWS lanzó Step Functions, el servicio del que forma parte el nuevo Workflow Studio.

Con Step Functions, los desarrolladores pueden combinar varios fragmentos de código Lambda en flujos de trabajo que realizan varias tareas y no solo una. Estos flujos de trabajo, a su vez, pueden ser utilizados por los desarrolladores para crear aplicaciones empresariales complejas, como servicios de procesamiento de pagos y herramientas de análisis. Sin embargo también es posible integrar funciones escritas en diferentes lenguajes de programación si utilizar estos servicios especializados [15].

Serverless cubre una amplia gama de tecnologías, que se pueden agrupar en dos categorías: Backend-as-a-Service (BaaS) y Functions-as-a-Service (FaaS).

Backend-as-a-Service permite reemplazar los componentes del lado del servidor con servicios listos para usar. BaaS permite a los desarrolladores externalizar todos los aspectos detrás de una escena de una aplicación para que los desarrolladores puedan elegir escribir y mantener toda la lógica de la aplicación en el frontend. Algunos ejemplos son los sistemas de autenticación remota, la administración de bases de datos, el almacenamiento en el cloud y el hosting.

Función como servicio es un entorno en el que es posible ejecutar software. Las aplicaciones serverless son sistemas basados en el Cloud impulsados por eventos donde el desarrollo de aplicaciones se basa únicamente en una combinación de servicios de terceros, lógica del lado cliente y llamadas a procedimientos remotos hospedados en el Cloud. [14]

Existen diversos desafíos, oportunidades y problemas a resolver, entre ellos la experiencia del desarrollador [17], Interoperabilidad, testing, composición de funciones, seguridad, administración del ciclo de vida, administración de requerimientos no funcionales, performance, optimización del overhead, ingeniería para costo-performance, entre otros [9]

Un diferenciador clave de serverless es la capacidad de escalar desde cero, o no cobrar a los clientes por el tiempo de inactividad. Escalar a cero, sin embargo, conduce al problema de los arranques en frío y el pago de la penalización de obtener código serverless listo para ejecutarse.

El inicio en frío se trata del retraso entre la ejecución de una función después de que alguien la invoque. Se trata de la función en el momento de la invocación. En background, FaaS utiliza contenedores para encapsular y ejecutar las funciones. Cuando un usuario invoca una función, FaaS mantiene el contenedor en ejecución durante un período de tiempo determinado después de la ejecución de la función (caliente) y si otra solicitud entra antes del apagado, la solicitud se sirve instantáneamente. El inicio en frío es aproximadamente el tiempo que se tarda en abrir una nueva instancia de contenedor cuando no hay contenedores en caliente disponibles para la solicitud.

También es importante entender que el bajo costo del servicio se debe a que los proveedores de FaaS no necesitan ejecutar la infraestructura en previsión del uso y pueden cerrar los recursos no utilizados.

Algunos usuarios que eligen FaaS, están aprovechando las mejoras de las plataformas, por ejemplo IBM está utilizando contenedores para reducir los arranques en frío y plataformas como OpenFaaS dan a los usuarios control sobre cómo quieren utilizar los recursos.

Aunque las plataformas serverless existentes funcionan bien para aplicaciones simples, no son adecuadas para servicios más complejos, especialmente cuando la lógica de la aplicación sigue una ruta de ejecución que abarca varias funciones [10].

2 Trabajos relacionados

No existen gran cantidad de trabajos que analicen la ejecución de funciones Serverless, con el objetivo de encontrar el modo más adecuado o eficiente de ejecutarlas. La mayoría de los trabajos muestras resultados sobre el arranque en frío y aspectos relacionados y estudiados desde diferentes ópticas.

El trabajo de Johannes Manner et al [12], presenta puntos de referencia económicos y también orientados al rendimiento. Compara las plataformas AWS y Azure, y los lenguajes Java y Javascript. Analizar los factores que influyen en la duración percibida del arranque en frío mediante la realización de un banco de pruebas en AWS Lambda y Microsoft Azure Functions con 49 500 ejecuciones de funciones Cloud. Su aporte más importante es con referencia a aspectos económicos y no hace propuestas de mejoras para el arranque en frío, sino que compara el comportamiento de las plataformas.

En el trabajo de David Bermbach et al [5], presenta tres enfoques (ingenuo, el extendido y la aproximación global), que reducen el número de arranques en frío durante el tratamiento del servicio FaaS como una caja negra, implementado como parte de un middleware coreográfico liviano, utilizando composición de funciones y por lo tanto, el aprovisionamiento de nuevos contenedores antes de que el proceso de aplicación invoque la función respectiva. Las pruebas las realizan sobre AWS Lambda y OpenWhisk, con un solo lenguaje de programación Node.js.

En la publicación, de Priscilla Benedetti et al [4], se explora la conveniencia de los modos de arranque en caliente y en frío para implementar aplicaciones de IoT, teniendo en cuenta un banco de pruebas de bajos recursos comparable a un nodo en el extremo (Edge). Modelando la implementación y el análisis experimental de una plataforma serverless que incluye elementos de servicio de IoT típicos. Presenta un estudio de rendimiento en términos de consumo de recursos y latencia y para realizar las pruebas utiliza OpenFaaS, un frameworks FaaS de código abierto que permite probar una implementación de arranque en frío con una configuración precisa del tiempo de inactividad gracias a su flexibilidad.

En la publicación [7], los autores describen los principales problemas que afectan al rendimiento de las plataformas serverless y presentan algunos resultados experimentales. Esta investigación hace uso de funciones disponibles comercialmente para tres estudios de casos específicos, siendo: el entrenamiento de modelos basado en aprendizaje automático, las predicciones en vivo y la clasificación de entradas de procesamiento por lotes. Los experimentos presentados son interesantes, pero no se analiza profundidad el problema del arranque en frío.

En el trabajo [13] se describen seis malas prácticas que han sido identificadas, y propone soluciones para tratar de superarlas. Las mismas son: las llamadas asincrónicas (que pueden incrementar la complejidad y requiere un canal de respuesta alternativo), las funciones que llaman a otras funciones (que causa una depuración compleja, y puede llevar un costo extra), el código compartido entre funciones (se podrían interrumpir las funciones serverless existentes que dependen del código compartido si éste cambia), el uso de demasiadas librerías (dado que se aumenta el espacio destinado a las mismas), adopción de demasiadas tecnologías como bibliotecas, frameworks, lenguajes (incrementa la complejidad del mantenimiento y aumenta los requisitos de conocimientos de los integrantes del proyecto) y demasiadas funciones que no son reusadas (causa menor mantenibilidad y menor comprensión del sistema).

En la publicación de Jiang, Choon Lee y Zomaya [11], aparecen resultados interesantes, a pesar de que dicho trabajo se enfoca en la ejecución de worksflow científico. Demuestran que FaaS ofrece un entorno de ejecución ideal para flujo de trabajo aplicado a las ciencias, con su mecanismo dinámico de asignación de recursos y un modelo de precios conveniente y además que AWS Lambda ofrece un entorno de ejecución ideal para aplicaciones de flujo de trabajo científicas con restricciones de precedencia complejas.

Un trabajo muy interesante por la descripción de los patrones de acceso de las funciones serverless desde IoT al Cloud fue desarrollado por [9]. Proponen el uso de WebAssembly como un método alternativo para ejecutar aplicaciones serverless y demuestran cómo una plataforma basada en WebAssembly proporciona muchas de las mismas garantías de aislamiento y rendimiento de las plataformas basadas en contenedores, al tiempo que reduce los tiempos medios de inicio de las aplicaciones y los recursos necesarios para hospedarlas.

3 Ejecución múltiple

Las funciones sin servidor no suelen implementarse de forma aislada. En su lugar, se desencadenan para realizar una tarea en respuesta a una acción o un evento. En una arquitectura distribuida típica, sirven como “pegamento” entre los diferentes componentes de la aplicación: el origen de un componente y el destino de otro. Como tal, es importante establecer el ámbito de los permisos asociados con una función siguiendo el "principio de privilegios mínimos".

La mayoría de los proveedores de FaaS tienen arranques en frío de 1 a 3 segundos y esto afecta a ciertos tipos de aplicaciones donde esta latencia tendrá un impacto dramático. El arranque en frío varía según el proveedor de la nube y los lenguajes de programación. Aunque tiene casi un año de antigüedad, este estudio de referencia muestra el impacto de la latencia de arranque en frío en varias ofertas de FaaS.

El objetivo de este trabajo es analizar cuál es la estrategia más adecuada para la ejecución múltiple de funciones y además cómo impacta el arranque en frio cuando se desea ejecutar múltiples funciones en AWS Lambda y si existen estrategias que puedan minimizar su impacto.

ISBN 978 -987-633-574-4

4 Desarrollo de la pruebas

Para realizar las pruebas se utilizó el servicio AWS Lambda. Este es un servicio informático serverless que permite ejecutar código sin aprovisionar ni administrar servidores, crear una lógica de escalado de clústeres basada en la carga de trabajo, mantener integraciones de eventos o administrar tiempos de ejecución. Es un servicio flexible y rentable que le permite implementar la funcionalidad de back-end en un entorno serverless [2].

Se tuvieron en cuenta los siguientes escenarios:

a) la ejecución Se realizaron las pruebas utilizando una función que utiliza hilos de ejecución escrita en Python, en la cual por cada hilo se invoca una sola función con 20, 50 y 100 invocaciones.

b) Se procedió a realizar las pruebas de ejecución múltiple con la función Lambda Invoke perteneciente a Lambda, la cual permite que una función Lambda llame a otra función Lambda, Se realizó con 20, 50 y 100 invocaciones.

La Función Invoke, invoca una función de Lambda. Puede invocar una función de forma sincrónica (y esperar la respuesta) o de forma asincrónica

Cabe aclarar que antes de poder ejecutar este tipo de funciones, se deben de definir un conjunto de permisos en los roles de ejecución que permita este tipo de invocación.

Se trató de encontrar si existía algún impacto del arranque en frío y por otro lado si la funcionalidad provista por Lambda (en este caso invoke) mejora la performance de la ejecución de instrucciones.

5 Resultados obtenidos

Para el caso de ejecución con hilos. Al utilizar 20 hilos un resultado de 2740,77ms en tiempo de ejecución, esto se muestra en la figura 1.

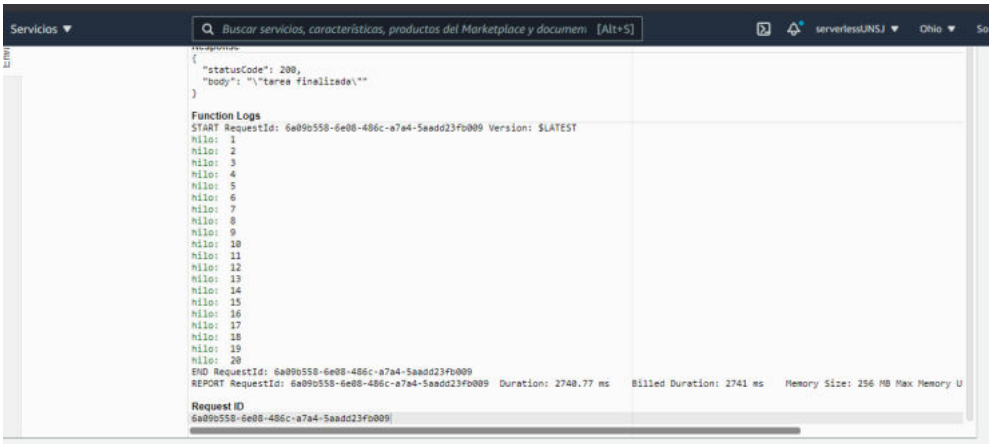

Figura 1

ISBN 978 -987-633-574-4

A continuación se probó invocando 50 funciones, donde se obtuvo un tiempo de 6112,13 ms. El tiempo obtenido es casi el doble al de la prueba anterior. Dichos resultados se muestran en la figura 2.

Figura 2

Para 100 funciones
Finalmente con 100 invocaciones de la función el resultado obtenido es 11594.42 ms, casi duplicando el tiempo obtenido anteriormente.
Como conclusión se puede afirmar que el tiempo de ejecución es proporcional a la cantidad de ejecuciones de la función, es decir, se puede equiparar a un orden de ejecución lineal, y no se observa impacto del arranque en frío
Los resultados se muestran en la figura 3

Figura 3

Para el caso b) (utilizando la función invoke):

ISBN 978 -987-633-574-4

Se partió de realizar la prueba para 20 funciones, la cual retorna un resultado de 1061,47ms en el tiempo de ejecución. Los resultados se muestran en la figura 4.

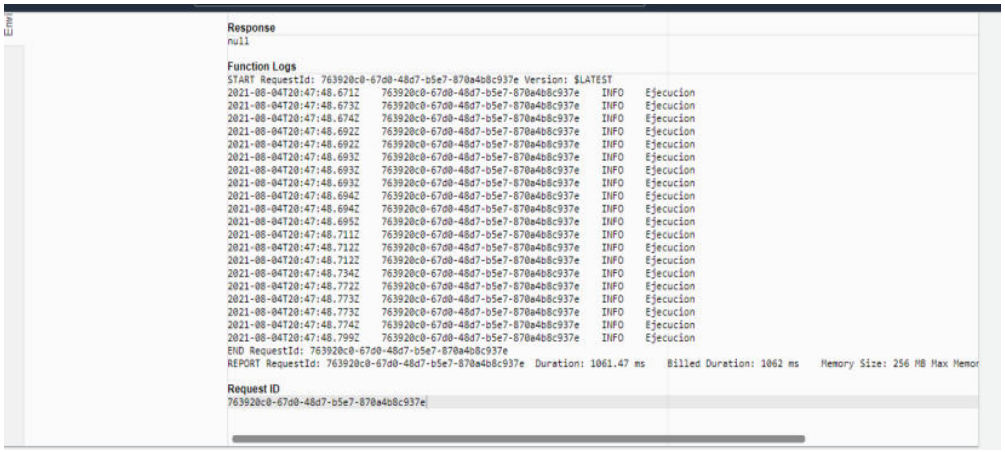

Figura 4

Luego se pasó a probar para 50 funciones, donde el tiempo aumentó proporcionalmente respecto a la prueba anterior, con un tiempo de ejecución de 1949,44 ms. Dicha ejecución se muestra en la figura 5.

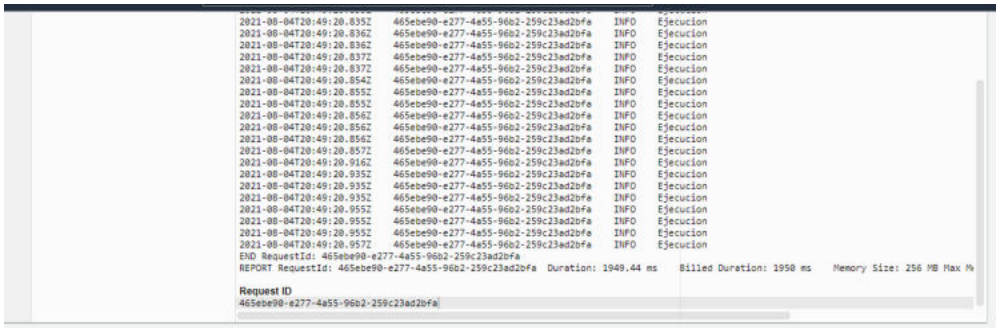

Figura 5

Por último se probó para 100 funciones, en este caso los resultados no fueron como se esperaba, ya que hubo una notable mejoría respecto del tiempo de ejecución de las anteriores pruebas, con un tiempo de ejecución de 2338,7 ms. La ejecución se muestra en la figura 6.

ISBN 978 -987-633-574-4

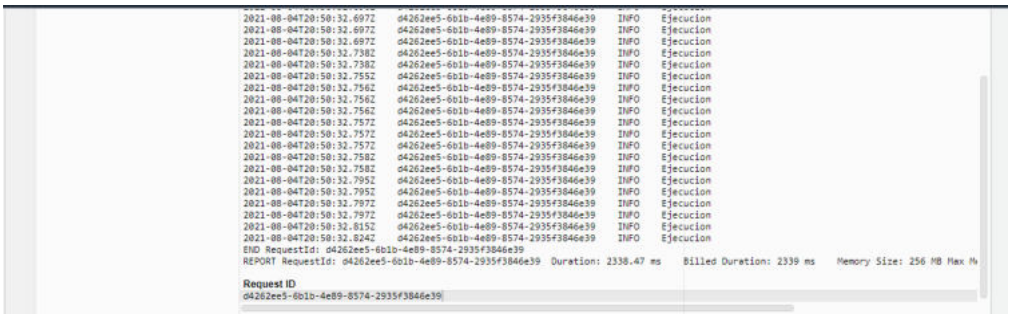

Figura 6

7 Conclusiones y Futuros trabajos

En función de los resultados obtenidos luego de realizar las seis pruebas, se puede concluir lo siguiente:

En ninguna de las pruebas se puede apreciar el impacto del arranque en frío, por lo tanto se considera que AWS provee recursos que minimizan este problema o al menos en estas pruebas no pudo apreciarse.

Por otro lado se debe considerar que Lambda Invoke es una buena alternativa a medida que se escala con la cantidad de funciones, ya que reduce la complejidad temporal lineal que se ve en la implementación con hilos.

Cabe aclarar además que los tiempos de ejecución utilizando Lambda Invoke han sido bastante menores respecto de la implementación con hilos, lo cual lo vuelve una propuesta tentadora a la hora de realizar ejecución múltiple de funciones.

Como trabajos futuros propuesto está el de aplicar concurrencia y paralelismo en la ejecución de funciones ya sea con Step Function o con alguna otra estrategia, evaluando si se mantiene la proporcionalidad en los tiempos de ejecución a medida que se escala en cantidad de funciones ejecutadas. Por otro lado, además se deberá realizar el análisis económico de la ejecución paralela (que dispondrá de más recursos por el escalado automático) que seguramente será más costoso.

También es de interés del grupo de investigación como continuación del presente trabajo, analizar los diferentes tipos de patrones de acceso de funciones serverless desde IoT, como son un solo cliente y múltiple acceso, múltiples clientes y un solo acceso y múltiples clientes, múltiples accesos.

Referencias

1. Adzic, G., Chatley, R.: Serverless computing: economic and architectural impact. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (pp. 884-889). ACM.(2917)
2. AWS Lambda: aws.amazon.com/es/lambda/

3. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V. & Suter, P.: Serverless computing: Current trends and open problems. In Research Advances in Cloud Computing (pp. 1-20). Springer, Singapore (2017).
4. Benedetti P. et al.: Experimental Analysis of the Application of Serverless Computing to IoT Platforms. Sensors (Basel, Switzerland) vol. 21,3 928. 30 Jan. 2021, doi:10.3390/s21030928
5. Bermbach D., Karakaya A., Buchholz S.: Using Application Knowledge to Reduce Cold Starts in FaaS Services. In: SAC '20, March 30-April 3, 2020, Brno, Czech Republic (2020).
6. Castro p., Ishakian v., Muthusamy v., Slominski a.: The rise of serverless computing. In: Communications of the ACM | Dec. 2019 | VOL. 62 | NO. 12 (2019).
7. Ekin Akkus I., Chen R., Rimac I., Stein M., Satzke K., Beck A., Aditya P., Hilt V.: SAND: Towards High-Performance Serverless Computing. In: 2018 USENIX Annual Technical Conference (2018).
8. Gottlieb, N. : State of the Serverless Community Survey Results.(2016) https: //serverless.com/blog/state-of-serverless-community/. (2016).
9. Hall A., Ramachandran U.: An Execution Model for Serverless Functions at the Edge. In: International Conference on Internet-of-Things Design and Implementation (IoTDI '19), April 15–18, 2019, Montreal,QC, Canada. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/ (2019).
10. Hellerstein, J.M.; Faleiro, J.M.; Gonzalez, J.E.; Schleier-Smith, J.; Sreekanti, V.; Tumanov, A.; Wu, C. Serverless Computing: One Step Forward, Two Steps Back. arXiv 2018, arXiv:1812.03651.
11. Jiang Q., Choon Lee, Zomaya A.: Serverless Execution of Scientific Workflows. In: Springer International Publishing. M. Maximilien et al. (Eds.): ICSOC 2017, LNCS 10601, pp. 706–721, 2017. https://doi.org/10.1007/978-3-319-69035-3_51 (2017).
12. Manner J., Endreb M., Heckel T., Wirtz G.: Cold Start Influencing Factors in Function as a Service. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (2018).
13. Nupponen Gofore J., Taibi D.: Serverless: What it Is, What to Do and What Not to Do. In: 2020 IEEE International Conference on Software Architecture Companion (ICSA-C) (2020).
14. Roberts M.: Serverless Architectures. https://martinfowler.com/articles/serverless.html (2016).
15. Rodríguez N., Atencio H. et al: Interoperabilidad de funciones en el Modelo de Programación de Serverless Computing. IV CICCSI. Universidad Champagnat (2020).
16. Serverless Architecture Market by Deployment Model, Application, Organization Size, and Industry Vertical. https://www.researchandmarkets.com/reports/4828585
17. Van Eyk1 E, Iosup A, Seif S., Thömmes M.: The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures. In: Proceedings of WoSC'17, Las Vegas, NV, USA, 4 pages. https://doi.org/10.475/123. (2017).