DOI: https://doi.org/10.15276/aait.02.2019.2
UDC 004.42: 004.8: 004.93

PRODUCTIVITY ESTIMATION OF SERVERLESS COMPUTING

Dmitry V. Kalnauz[1)]
ORCID: https://orcid.org/0000-0002-8042-1790, dmitrysavchyk@gmail.com
Viktor A. Speranskiy[1)]
ORCID: https://orcid.org/0000-0002-8042-1790, speranskiyva@ukr.net
[1)] Odessa National Polytechnic University, 1, Shevchenko Avenue. Odesa, 65044, Ukraine

ABSTRACT

Cloud computing has enabled organizations to focus less on their IT infrastructure and more on their core products and services. In fact, Cloud is no longer viewed as an alternative to hosting infrastructure. Serverless computing is a technology, also known as function-as-a-service, that gives the cloud provider complete management over the container function run on as necessary to serve requests. As a result, the architectures remove the need for continuously running systems and serve as event driven computing. Serverless computing presents new opportunities to architects and developers of Cloud-oriented solutions. Primarily, it provides a simplified programming model for distributed Cloud-based systems development, with the infrastructure abstracted away. It is no longer the concern of the developer to manage load balancers, provisioning and resource allocation (although system implementers need to be aware of such things). This reduced focus on operational concerns should allow greater attention to be paid to delivering value, functionality and an ability to adapt rapidly to changes. Such issues as deployment, monitoring, quality of service and fault tolerance are moved into the hands of the Cloud provider and still need to be actively considered and managed. Serverless computing is still in its infancy and while the model matures further, tools will be created to allow developers and architects to create patterns and processes to fully exploit the advantages of the Serverless model. This paper explores the performance profile of a Serverless ecosystem under low latency and high availability. The results of application and performance tests for image recognition by using neural networks are presented. The proposed implementation uses open source libraries and tools: TensorFlow for the study of machine learning and LabelImg for data preparation. A correlation between the amount of experimental training data and recognition accuracy is studied and shown. For experiments, the software package was developed using the Python scripting programming language and .Net technology. The developed software showed excellent accuracy of recognition using regular computer with low-cost hardware. Interaction of the client side with the "server" is carried out using HTTP-requests in any browser with low-speed network connection.

Keywords: Serverless; Cloud Computing; FaaS; Amazon Web Services Lambda; Microsoft Azure Cloud Function; Google Cloud Platform Functions

***For citation*: Kalnauz D. V., Speranskiy V. A. Productivity Estimation of Serverless Computing. *Applied Aspects of Information Technology*. 2019; Vol.2 No.1:20–28. DOI: https://doi.org/10.15276/aait.02.2019.2**

INTRODUCTION

Serverless is a leading technology, since it working physically on a server, but it does not need to configure infrastructure. Serverless can be distinguished among such an event-oriented architecture and function as a service (Function-as-a-Service) [1]. We can see that the Serverless architecture offers application computing for the microservices in which the event is caused by other systems and resources, and the micro-services are described as formal syntax written in program functions. A new entry in the database, repository allocation, or Internet notifications is a variety of examples of events that may simply be messages or will be processed. Sometimes an event is created with a certain amount of time with a subscription, but in many cases, a significant amount of event messages must be processed immediately. Horizontal scaling for processing simultaneous queries is one for the characteristics of cloud computing [2]. New event message handled in an instance of the isolation function and few examiners are needed when several event messages are created simultaneously.

The event created by mobile application, processed with light weights, but the amount of incoming traffic is usually unpredictable, so such programs must be deployed on a specific platform, build with using dynamic redundancy and resource management, such as Serverless computing [3].

On the one hand, it provides developers with a simplified programming model for creating cloud applications, which eliminates most, if not all, operational problems; it reduces the cost of deploying cloud code by charging for execution time, rather than for resource allocation; and this is a platform for rapid deployment of small pieces of cloudy code.

Serverless model provides new capabilities that make writing more scalable microservices easier and cost effective as the next step in evolution of cloud computing architectures that can be used for different technology tasks. There is a series of tasks devoted to development of easy and effective solutions with use of modern cloud functions. However, most of them cannot be tested using regular low-cost equipment.

The aim of the work is to estimate the productivity of Serverless computing for image recognition tasks. To attain the aim, it is needed to solve the next tasks: perform a review of the modern cloud compu-

ISSN 2617-4316 (Print)
ISSN 2663-7723(Online)

ting technologies and develop corresponding software tools, that can be used both on regular PC and cloud platform.

1. TECHNOLOGY INTRODUCTION

1.1. State of art of Serverless Computing

Someone thinks that servers are not needed for Serverless computing [4]. This is actually not true. Serverless functions still use physical servers. To explain this, we use an example of a traditional n-tier application with server logic and show how it will differ using Serverless architecture (Fig. 1).

In a Serverless architecture, several things can change including the server and the database. An example of this change would be creating a cloud-provisioned API and mapping specific method requests to different functions.

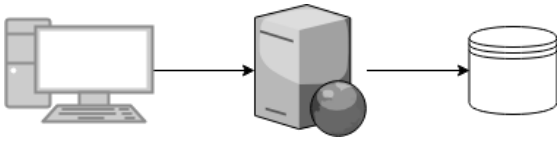

Fig. 1. **Traditional architecture in which server provide and managed by developer**
Source: **compiled by the authors**

Instead of having one server, our application now has functions for each piece of functionality and cloud-provisioned servers that are created based on demand. We could have a function for searching for a book, and a function for purchasing a book. We also might choose to split our database into two separate databases that correspond to the two functions (Fig. 2).

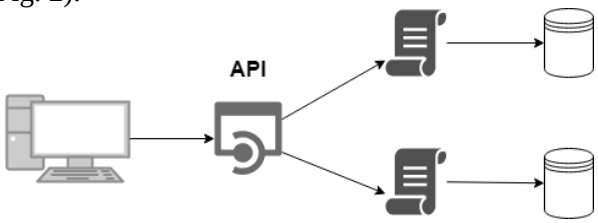

Fig. 2. **Serverless architecture where servers are scale up and down based on demand**
Source: **compiled by the authors**

There are a couple of differences between the two architecture diagrams. One is that in the on-premises example, you have one server that needs to be load-balanced and auto-scaled by the developer. In the cloud solution, the application is run in stateless compute containers that are brought up and down by triggered functions. Another difference is the separation of services in the Serverless example [5].

Triggers are simply events. They are services and HTTP requests that create events to start up functions for response. Triggers are usually set within the function console or the command-line interface and are typically created within the same cloud provider's environment. A function must have exactly one trigger. There are three types of triggers: HTTP trigger, Database trigger and Object Storage trigger.

1) HTTP Trigger is a simple but provide rich format for call function with various content type, such as a files, text, JSON, and PUT, POST and DELETE HTTP methods.

2) Database Trigger call function when there is an insertion, modification or deletion of any record in a table, which behaves like a stack collection. Google provide pub/sub trigger in Serverless platform and it would be exchangeable by database trigger because Google Function does not have database trigger.

3) Storage Object Trigger.

In AWS, a trigger can be an HTTP request or a call to another AWS service. Azure functions also use service triggers, but they also capture the idea of bindings. Input and output bindings offer a declarative way to connect to the data of your code. Bindings are not similar to triggers, as you, as a developer, specify connection strings and other properties in your configuration functions. Unlike launching, bindings are optional, and a function can have multiple bindings.

An example of a program with a trigger is the record in the API Query Tab. We have a table in Azure storing information about employees and whenever a POST request comes with new information about employee and we want add another row in the table. We can do this by running the HTTP Trigger, the Azure function, and the Tabbed output bindings.

By using the trigger and bindings, we can write more general code, which does not make the function of relying on the details of the services with which it interacts. Information about incoming events from services is introduced into our function. Data output to another service, for example, adding a row to a table in the Azure tables' repository, may be the execution of using the value that returns to our Function. The Trigger and HTTP bindings have the name of the authority, Please Act as an Identifier that will be used in the Functions Code to access the trigger and accessory. The trigger and bindings can be configured on the Azure Functions portal integration tab. This configuration is displayed in the function JSON file in the function directory. This file can also be manually configured in Extension Editor.

Serverless computing calls can support distributed data processing with bandwidth, latency, and distributed computing performance. There are certain limitations that we need to know before using the function, for example, there are several event handlers:

1) HTTP, Object Storage and Database;

2) Not large amount of memory – from 512 MB to 3 GB of memory per container;

3) Maximum time allowed for the function is allowed from 5 minutes to 10 minutes;
4) 500 MB cache.

Platforms comparison could be helpful for new users of Serverless and may to understand the base information of the Serverless platform (Table 1) [6].

Amazon Lambda was the first Serverless platform that was presented in 2014 [7]. It defined few key aspects like a cost, programming model, deploy, security and monitoring. That supports many languages, e.g., Node.JS, Python, Java, GoLang, .NET. Platform use advantage of AWS's ecosystem [8].

Microsoft Azure Functions provide HTTP webhooks and integration with Microsoft Azure web services. The platform supports C#, F#, Node.JS, TypeScript, Batch, Bash, PowerShell and Java. The runtime code is open-sourced and available on GitHub repository under MIT license [9], [14].

Google Cloud Functions provides basic functions to run Serverless functions that wat written in Node.JS for HTTP calls or events from another Google Cloud services. The functionality currently is limited but expected grow in future [10].

Table 1. **Platform comparison**

	AWS Lambda	Google Functions	Azure Functions
Programming language	Node.js, Python, Java, NET, Golang	Node.js	C#, F#, Node.js, PHP, TypeScript, Batch, Bash, PowerShell, Java
Triggers	18 triggers (with S3, DynamoDB)	3 triggers	6 triggers (with Blob, Cosmos DB)
Memory price	$0.0000166/GB-s	$0.00000165/GB-s	$0.000016/GB-s
Execution price	$0.2 per 1M	$0.4 per 1M	$0.2 per 1M
Free Tier	First 1M	First 2M	First 1M
Maximum memory	3008MB	2048MB	1536MB
Operation system(OS)	Linux	Debian GNU/Linux 8 (jessie)	Windows NT
CPU per container	2900 MHz,1 core	1.4GHZ	2200 MHz, 2 Processors
Maximum code size	50/250MB (compressed/ uncompressed)	100/500MB (compressed/ uncompressed)	100/500MB (compressed/ uncompressed)

Source: **compiled by the authors**

According to the table, AWS Lambda offers a widest range of programming language [11]. We also could see that cost price based on metrics, first – the number of invocations by function. Second, the time that is taken by a function to execute. Invocation to the Serverless function is cost-effective in all Serverless providers. All providers have similar price policy.

Each Serverless platform provides different programming **language support**, which developers can use for creating function with their own a language preference [12]. As interpreted language, we can find Node.js for JavaScript and Python runtime environment, as most supported. Compiled languages such as Java and .NET are also supported, although there is no built-in web editor for their languages. The Table 2 shows the languages supported by each platform.

Table 2. **Language support comparison**

Programming Language	AWS	Google	Azure
Python	2.7, 3.6	2.7	-
Java	8	-	8
NodeJs	4.3, 6.10, 8.10	6.11, 5	6
NET Core	1, 2	-	1, 2
Other	Golang 1.x	-	F# 4.6, Experimental(Python, PHP Batch, Bash, PowerShell)

Source: **compiled by the authors**

1.2. Evaluation of Serverless v.s.Virtual Machine

Serverless does not offer high performance computing or a cheap pricing model compared to Amazon EC2. Virtual machines in cloud computing offers several options for scaling computing resources, through network bandwidth and performance, which requires optimal planning, and management. Serverless provide resource processing for lightweight functions without management objectives and offer cost-effective solutions.

Amazon, for example, offer a wide range of EC2 machines optimized for various task and reaches 128 CPU and 3.8TB of memory. AWS Lambda provide to launch function thousandth time with small amount of memory (up to 3008MB or 2.8GB), which can reach to 2.8TB.

Serverless is powered by containers, which have near zero start-up and run without latency during a function life cycle.

For this comparison we should use function that requested allocate CPU resources to an instance of a function with simultaneous calls [13]. Multiplying for two-dimension array (matrix) is suitable for this. Fig. 1 shows the function execution time with multiplying 50; 100; 250; 500 and 1000 elements in each array.

For each case, several dozens of launches were carried out to avoid different nature delays and other actions that could lead to errors in the results. The Fig. 3 shows the averaged results. It can be noticed that the AWS lambda has a slight performance ad-

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

vantage in compare to Google Cloud and Microsoft Azure Functions.

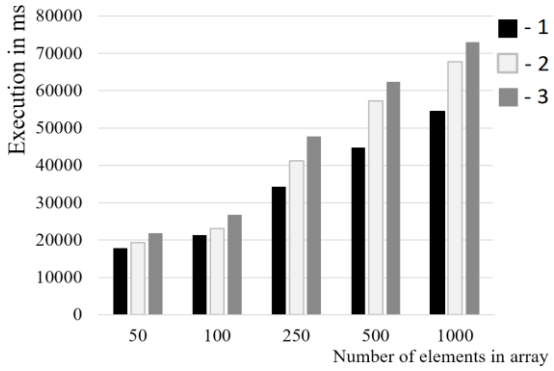

Fig. 3. **Dependency of execution time on array length for different cloud computing providers: 1 – Amazon; 2 – Microsoft; 3 – Google**
Source: **compiled by the authors**

Multiplication of multidimensional arrays requires considerable resources. The performance result of multiplying of two 500 elements arrays followed by function calls is presented in Fig. 4.

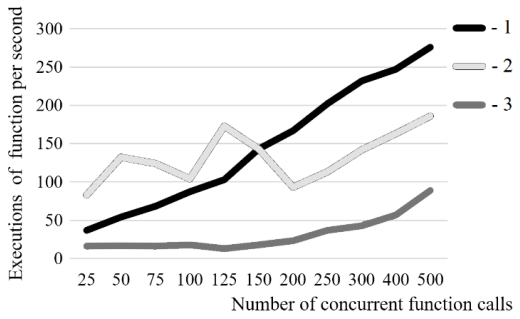

Fig. 4. **Function bandwidth with concurrent calls: 1 – Amazon; 2 – Microsoft; 3 – Google**
Source: **compiled by the authors**

The measurements were carried out with different numbers of simultaneous calls, from 25 to 500. AWS showed an almost linear relationship during the call and the worst result with scalability from all platforms. Measurements were carried out from all platforms. The performance of Azure features is very different on other platforms with fewer calls. Interestingly, it persists throughout the iterations of the test. At the same time, it showed almost lazy dependence with such calls and better results with small numbers of calls.

2. PRACTICAL USE OF TECHNOLOGY

There are several areas where Serverless can play an important role as in research as well as in a commercial using. Image or document processing for CDN is applicable for Serverless. Internet of Things (IoT) is also one of the use cases for Serverless, because IoT devices typically have a small computing power to process information and they need to user remote processing resources. For example, there is cooling and another similar process that requires constant temperature control. When cooling is not working or there are problems with work, function can execute live migration of workload and/or send signal about problem.

Advantages:

1) Cost: Serverless can be more cost-effective than renting or purchasing a fixed quantity of servers which generally involves significant periods of underutilization or idle time. It can even be more cost-efficient than provision.

2) Elasticity: in addition, a Serverless means that developers and operator do not need to spend time for setting up auto scaling or systems. The cloud provider is responsible for seamlessly scaling the capacity to the demand.

3) Small teams of developers are able to run code themselves without the dependence upon teams of infrastructure and support engineers; more developers are becoming DevOps skilled and distinctions between being a software developer or hardware engineer are blurring.

4) Productivity: one of the greater benefits in implementing a Serverless solution in its ease of use. There is little ramp-up time need to begin programming for a Serverless application. Most of this simplicity is thanks to services, provided by cloud providers that make it easier to implement solutions. The programmer does not need to implement or work with multithreading or handling HTTP requests in their code.

Disadvantages:

1) Performance: Serverless may suffer greater response latency than code that is continuously running on a dedicated server, virtual machine. This is because cloud providers typically “pull down” the Serverless code completely when not in use. This means that if the runtime (such as Java and .Net runtimes) requires amount of time to start up – it create additional latency.

2) Resource limits: Serverless computing is not suited to some computing workloads, such as high-performance computing, because of the resource limits imposed by cloud providers, and because it would likely be cheaper to bulk-provision the number of servers believed to be required at any given point in time.

3) Monitoring and debugging: diagnosing performance or excessive resource usage problems with Serverless code may be more difficult than with traditional server code, because although entire functions can be timed, there is typically no ability to dig

into more detail by attaching profilers, debuggers or APM tools. Furthermore, the environment in which the code runs is typically not open source, so its performance characteristics cannot be precisely replicated in a local environment.

4) Standards: Serverless computing is very new and not currently bounded by standards so that portability can be an issue when moving business logic from one public cloud to another. Cloud Native Computing Foundation (CNCF) is working on developing a specification with Oracle.

2.1. Optimization

Circuit Breaker Pattern allows a call to an unresponsive system component to be aborted without needlessly consuming resources trying to repeatedly connect and retry. There will be occasions when components are unresponsive and the system should be able to handle this without cascading failure. It is in situations like this that retry is not beneficial and may well have harmful effects if it ends up spinning up many cold Lambdas. A circuit breaker is required that will identify when a system is in stress and will back off. If this is linked with the front-end it would become possible for the server to issue a 503 HTTP response and the front-end to silently retry after a predetermined back-off.

Bulkhead pattern effectively isolate components of the system that display inconsistent latency. These may be Lambdas, which take a variable amount of time to complete based on the workload or which interact with external systems with an inconsistent performance profile.

As an example, the case study project initially had a single function, which handled customer user data. The architecture was such, that customer data was refreshed and cached from an external system during the initial authentication process. However, other functions, which needed access to the customer data frequently, would call the customer requesting the cached data. The asymmetrical nature of the performance profile between the refresh and request calls, with the refresh operation suffering significantly higher latency than the request call, could cause refresh calls to unnecessarily divert requests to cold Lambdas.

Implementing bulkheads separating high latency operations from low latency application request flows significantly reduced the probability of a given request being impacted by cold functions. In real terms this required separating the request and refresh functionality into separate functions to prevent high latency in one part of the system adversely affecting another.

Appropriate language to Serverless development significantly improves latency within some parts of the system.

Teams responsible for Lambda development should use the language best suited to the particular service. Whilst this may reduce code reusability, it allows for a reduction in latency in system components that are highly sensitive to AWS Lambda initialization timings. Using either Node.js or Python Lambdas on front-end facing Lambdas reduces latencies since these languages are less susceptible to problems with cold starts and can then offload to Lambdas implemented in other languages in a manner, which would not negatively affect the user experience.

As stated as part of the AWS Lambda best practice documentation “the compiled languages (Java and .NET) incur the largest initial start-up cost for a container’s first invocation, but show the best performance for subsequent invocations [14]. The interpreted languages (Node.js and Python) have very fast initial invocation times compared to the compiled languages, but can’t reach the same level of maximum performance”. The implication is that latency sensitive applications or those expecting spiky traffic should use interpreted runtimes where possible. It can be further extrapolated that Lambdas forming part of the same application can use different runtimes depending on the predicted workload for a specific component.

The warming strategy using functions – is a final way to increase productivity that can be used with the approaches outlined above. It is used to ensure that the appropriate amount of function is always warm. This approach protects a scheduled function implementation that makes bogus calls to other function s in the system so that they are forced to keep warm. This approach imposes some optimization requirements, since it must be predetermined, which and how much function should be kept warm. Although this approach somewhat weakens the goal of the system, which should dynamically scale in response to demand, it is nevertheless a viable strategy to mitigate the effects of cold function on overall latency. It should be noted, that this approach will not lead to a significant increase in deployment costs, since calls to test communication with the corresponding functions should be rarely performed, but at some point, it is difficult to predict the requirements for using functions. In this regard, there may be a problem associated with large bursts of demand, but it can reduce problems when used in combination with other methods described in this section.

2.2. Experimental steps

TensorFlow is an open library for machine learning research and product development, built by Google for learning neural networks [15]. TensorFlow offers an API for development for personal computers, mobile devices, the web and cloud computing. TensorFlow neural networks are expressed as a state of data flow graphs. Each node in the

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

graph represents operations performed using neural networks on multidimensional arrays. TensorFlow architecture allows deploying at multiple processors or graphics processors within the desktop, server or mobile device.

Before sending data for network training, they are pre-processed using "training with a teacher" [16]. This process consists of labeling images. This is one of the most time-consuming tasks in data preparation. For this, the freely distributed Labelimg tool (graphical image annotation tool) was used [17], which automatically creates an XML file with the coordinates of the marked objects in the photo (Fig. 5).

Fig. 5. **Objects markup – light rectangles with bold points in the corners showing selected Images**
Source: **compiled by the authors**

After completing the marking of images, we proceed to the next task, which separates the test data. It is common practice to divide the indicated data into a training and test set.

The model is trained in the training set, and its performance is checked for how well it summarizes the data that have never been seen before in the test set. The performance of the model on the test suite gives an idea of how the model works, and allows you to solve problems such as compromises of lead and deviations. The general rule is to deploy 90% of the data on the training set, and the remaining 10% on testing randomly.

The TensorFlow Object Detection API was used. To continue, it is needed to select the model to be trained. The *ssdlite_movilenet_v2_coco* was chosen for high performance required for work with streaming video.

The training procedure lasted about 6 hours on a regular computer. For more convenience, the data were divided into two categories: photos and videos. At some point, the process was stopped to check the results with TensorBoard.

The most important and most valuable is the metric of total spending: the smaller the loss, the better the module is executed. Losses are calculated both on the training kit and on the test kit (Fig. 6), as well as on the interpretation of how well the model performs on the two sets.

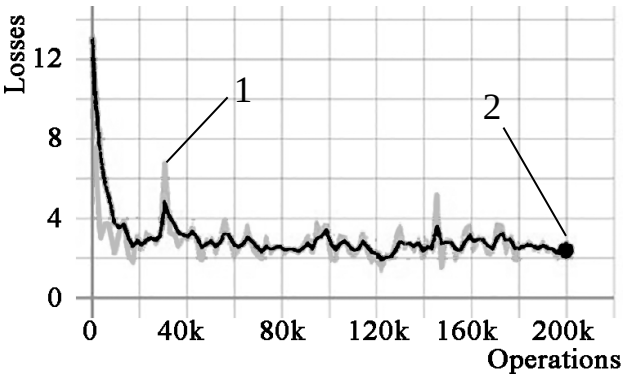

Fig. 6. **Total losses:**
1 – errors in training over training steps;
2 – after some smoothing
Source: **compiled by the authors**

Losses are estimated not in percentages (unlike precision), but as the sum of errors made for each example in sets for training or testing. When data is smoothed (regularization), there is an increase in losses depending on the number of training steps (Fig. 7).

In the case of neural networks, the loss is usually negative likelihood (mainly cross-entropy) or residual sum of squares (or the sum of squares of prediction errors) for classification and regression, respectively. Then, the main goal in the training model is to reduce (minimize) the values of the loss function with respect to the model parameters by changing the values of the weight vectors using various optimization methods.

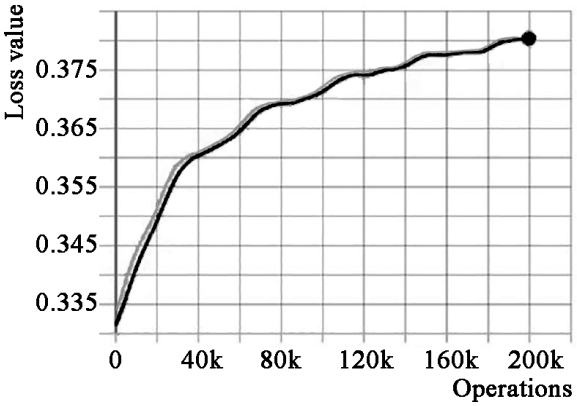

Fig. 7. **Losses after data regularization**
Source: **compiled by the authors**

For example, back distribution in neural networks. The loss value means how well or poorly defined a model behaves after each iteration of the

optimization. Ideally, one would expect a decrease in losses after each or several iterations.

The accuracy of the model, as a rule, is determined after studying and fixing the parameters of the model and the absence of training. Then, test cases are served on the model and the number of errors (zero loss) that the model allows is fixed after comparing with the real goals (Fig. 8). Then the percentage of misclassification is calculated.

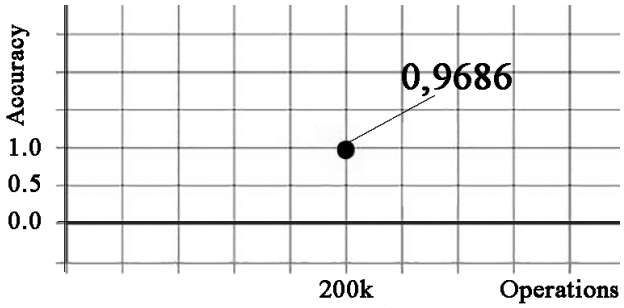

Fig. 8. **The identification accuracy of developed software tools**
Source: **compiled by the authors**

For the equipment of our experiment (without using a graphics processor), 1000 training events took about 3 hours, and 2000 training stages took about 6 hours. Most of the training procedure ends after 3 hours, and in the last hour of the experiment, there was no real improvement, so it was decided to stop the learning process.

3. CONCLUSIONS

Function based on Serverless computing can process distributed data applications and provide quick access to additional compute resources. Serverless computing is an event-driven FaaS technology that utilizes third-party technology and servers to remove the problem of having to build and maintain infrastructure to create an application.

Overall, Serverless computing can be used for distributed date computing, if divided task is small to perform with 1.5-3 GB memory restriction and execution time up to 15 minutes. From this we can conclude that Serverless computing is more cost-effective than processing with traditional virtual machines because almost zero delay on boot up new instances and a charging model only for the execution time of function instead of paying for an idle time of machines.

Nowadays, Serverless computing uses containers with small amount of computing resources. We can conclude that in the future there will be more functional features with fewer configurations. They will be used for solving complex and resource-intensive computing.

The developed software toolkit showed very high accuracy of recognition (0.9686) after continuous 3 hour training using regular PC based on low-cost hardware equipment.

Current experiments showed that growing of data quantity and time needed for experiment does not leads the increasing of recognition accuracy using *TensorFlow* library in Serverless implementation.

Experimenting with training of other network models to reduce the time and improve the accuracy of recognition is the subject of further research.

REFERENCES

1. Wang, L., Li, M., Zhang, Y., Ristenpart, T. & Swift, M. “Peeking behind the curtains of Serverless platforms”. Proceedings of USENIX Annual Technical Conference (USENIX ATC’18). June 2018. p. 133–146.

2. 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. Publ. Springer*. Singapore: 2017. p.1–20.

3. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q. & Gonzalez, J. E. “Cloud Programming Simplified: A Berkeley View on Serverless Computing”. *arXiv preprint arXiv:1902.03383*. 2019.

4. Frazer Jamieson, Losing the server? – Available from: – *https://www.bcs.org/content/conWebDoc/58491*. Retrieved – March 1. 2019.

5. McGrath, G. & Brenner, P. R. “Serverless computing: Design, implementation, and performance”. *In 2017 IEEE 37-th International Conference on Distributed Computing Systems Workshops (ICDCSW).* June 2017. p. 405–410. IEEE.

H. Lee, K. Satyam & G. Fox. “Evaluation of Production Serverless Computing Environments”. *In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD).* San Francisco: CA, USA. 2018. p. 442–450. DOI: https://doi.org/ 10.1109/CLOUD.2018.00062.

6. “Introducing AWS Lambda”. Available from: https://aws.amazon.com/ru/about-aws/whats-new/2014/11/13/introducing-aws-lambda. – Retrieved March 1. 2019.

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

7. Hegde, M., Petrenko, M., Smit, C., Zhang, H., Pilone, P., Zasorin, A. A., & Pham, L. “Giovanni in the Cloud: Earth Science Data Exploration in Amazon Web Services”. *In AGU Fall Meeting Abstracts*. December 2017. p.17–24,

8. “Ron Miller, Microsoft answers AWS Lambda’s event-triggered Serverless apps with Azure Functions. – Available from: https://techcrunch.com/2016/03/31/microsoft-answers-aws-lambdas-event-triggered-serverless-apps-with-azure-functions/. – Retrieved March 1. 2019.

Malawski, M., Gajek, A., Zima, A., Balis, B., & Figiela, K. (2017). “Serverless execution of scientific workflows: Experiments with HyperFlow”, AWS lambda and Google cloud functions. Future Generation Computer Systems. p.1–15. DOI: https://doi.org/10.1016/j.future. 2017.10.029.

9. Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C. & Arpaci-Dusseau, R. H. “Serverless computation with openlambda”. *In 8-th USENIX Workshop on Hot Topics in Cloud Computing* (HotCloud 16). 2016. p.1–7.

10. Ghodsi, A., Shankar, S., Paranjpye, S., Xin, S. & Zaharia, M. “Serverless execution of code using cluster resources”. U.S. Patent Application. 2018; No. 15/581: 987.

11. Geng, X., Ma, O., Pei, Y., Xu, Z., Zeng, W. & Zou, J. “Research on Early Warning System of Power Network Overloading Under Serverless Architecture”. *In 2018 2-nd IEEE Conference on Energy Internet and Energy System Integration (EI2).* October 2018. p. 1–6. IEEE.

12. Rosenbaum, S. “Serverless computing in Azure with. NET”. *Packt Publishing*. 2017. 468 p.

13. Géron, Aurélien. “Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems”. *O'Reilly Media, Inc.*

14. Ao, L., Izhikevich, L., Voelker, G. M. & Porter, G. “Sprocket: A Serverless Video Processing Framework”. In Proceedings of the ACM Symposium on Cloud Computing. October 2018. p. 263–274. ACM.

15. Prentice, C., & Karakonstantis, G. “Smart Office System with Face Detection at the Edge”. *In 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDC om/IOP/SCI)*. October 2018. p. 88–93. IEEE.

Conflicts of Interest: the authors declare no conflict of interest

Received 20.12.2018
Received after revision 14.02.2019
Accepted 20.02.2019

DOI: https://doi.org/10.15276/aait.02.2019.2
УДК 004.42: 004.8: 004.93

ОЦІНКА ПРОДУКТИВНОСТІ БЕЗСЕРВЕРНІХ ОБЧИСЛЕНЬ

Дмитро Валерійович Калнауз[1]
ORCID: https://orcid.org/0000-0002-9970-6833, dmitrysavchyk@gmail.com
Віктор Олександрович Сперанський[1)]
ORCID: https://orcid.org/ 0000-0002-8042-1790, speranskiyva@ukr.net
[1)] Одеський національный політехнічний університет, пр-т Шевченка, 1. Одеса, 65044, Україна

АНОТАЦІЯ

Хмарні обчислення дозволили організаціям менше зосередитися на своїй IT-інфраструктурі і більше на своїх основних продуктах і послугах. Serverless – це технологія, також відома як функція-як-послуга, яка за необхідності надає постачальнику послуг хмарних обчислень повний контроль над контейнером для обслуговування запитів, на якому виконується функція. Як наслідок, архітектури виключають необхідність постійно працюючих систем і слугують обчислювальним процесом, керованим подіями. Serverless-обчислення відкривають нові можливості для архітекторів та розробників, орієнтованих на хмарні обчислення. Вона забезпечує спрощену модель програмування для розробки розподілених Cloud-систем, з відстороненою інфраструктурою. Serverless обчислення все ще перебувають у зародковому стані та з подальшим розвитком моделі будуть створені інструменти, що дозволять розробникам і архітекторам створювати моделі та процеси, щоб більш повно використовувати переваги моделі Serverless. У даній роботі розглянуто профіль продуктивності Serverless екосистеми в умовах низьких затримок і високої доступності. Представлено результати застосування і тести продуктивності для розпізнавання образів з використанням нейронних мереж. У реалізації використовуються відкриті бібліотеки та інструменти: TensorFlow для вивчення машинного навчання і LabelImg для підготовки даних. Показана кореляція між кількістю експериментальних навчальних даних і точністю розпізнавання. Для експериментів був розроблений програмний пакет з використанням скриптової мови програмування Python і технології .Net. Розроблене програмне забезпечення показало відмінну

точність розпізнавання використовуючи звичайний комп'ютер з недорогим обладнанням. Взаємодія клієнтської сторони з «сервером» здійснюється за допомогою HTTP-запитів.

Ключові слова: Serverless; хмарні обчислення; функція-як-послуга; Amazon Web Services Lambda; Microsoft Azure Cloud Function; Google Cloud Platform Functions

DOI: https://doi.org/10.15276/aait.02.2019.2
УДК 004.42: 004.8: 004.93

ОЦЕНКА ПРОДУКТИВНОСТИ БЕССЕРВЕРНЫХ ВЫЧИСЛЕНИЙ

Дмитрий Валерьевич Калнауз[1)]
ORCID: https://orcid.org/ 0000-0002-9970-6833, dmitrysavchyk@gmail.com
Виктор Александрович Сперанский[1)]
ORCID: https://orcid.org/0000-0002-8042-1790, speranskiyva@ukr.net
[1)] Одесский национальний политехнический университет, пр-т Шевченко, 1. Одесса, 65044, Украина

АННОТАЦИЯ

Облачные вычисления позволили организациям меньше сосредоточиться на своей ИТ-инфраструктуре и более на своих основных продуктах и услугах. Serverless -– это технология, также известная как функция-как услуга, при необходимости предоставляет поставщику услуг облачных вычислений полный контроль над контейнером для обслуживания запросов, на котором выполняется функция. Как следствие, архитектуры исключают необходимость постоянно работающих систем и служат вычислительным процессом, управляемым событиями. Serverless-вычисления открывают новые возможности для архитекторов и разработчиков, ориентированных на облачные вычисления. Она обеспечивает упрощенную модель программирования для разработки распределенных Cloud-систем, с отстраненной инфраструктурой. Serverless вычисления все еще находятся в зачаточном состоянии и с дальнейшим развитием модели будут созданы инструменты, которые позволят разработчикам и архитекторам создавать модели и процессы, более полно использовать преимущества модели Serverless. В данной работе рассмотрен профиль производительности Serverless экосистемы в условиях низких задержек и высокой доступности. Представлены результаты применения и тесты производительности для распознавания изображений с использованием нейронных сетей. В реализации используются открытые библиотеки и инструменты: TensorFlow для изучения машинного обучения и LabelImg для подготовки данных. Показана корреляция между количеством экспериментальных обучающих данных и точностью распознавания. Для экспериментов был разработан программный пакет с использованием скриптового языка программирования Python и технологии. Net. Разработанное программное обеспечение показало отличную точность распознавания, используя обычный компьютер с недорогим оборудованием. Взаимодействие клиентской стороны с «сервером» осуществляется с помощью HTTP-запросов.

Ключевые слова: Serverless; облачные вычисления; функция как услуга; Amazon Web Services Lambda; Microsoft Azure Cloud Function; Google Cloud Platform Functions

ABOUT THE AUTHORS

Dmitry V. Kalnauz, Department of Computerized Control Systems, Odessa National Polytechnic University, 1, Shevchenko Avenue. Odesa, 65044, Ukraine
dmitrysavchyk@gmail.com. ORCID: https://orcid.org/ 0000-0002-8042-1790

Калнауз, Дмитро Валерійович, каф. Комп'ютеризованих систем управління.. Одеський національный політехнічний університет, пр-т Шевченка, 1. Одеса, 65044, Україна

Viktor A. Speranskiy, PhD (Eng), Associate Professor, Associate Professor at the Department of Computerized Control Systems, Odessa National Polytechnic University, 1, Shevchenko Avenue. Odesa, 65044, Ukraine
speranskiyva@ukr.net.ORCID: https://orcid.org/0000-0002-8042-1790.

Віктор Олександрович Сперанський, кандидат техніч. наук, доцент каф. Комп'ютеризованих систем управління.Одеський національний політехнічний університет, пр-т Шевченка, 1. Одеса, 65044, Україна