Adaptation of Workflow Application Scheduling Algorithm to Serverless Infrastructure.

Maciej Pawlik✉, Pawel Banach, and Maciej Malawski

AGH University of Science and Technology, Krakow, Poland
m.pawlik@cyfronet.pl, pbanach@student.agh.edu.pl, malawski@agh.edu.pl

Abstract. Function-as-a-Service is a novel type of cloud service used for creating distributed applications and utilizing computing resources. Application developer supplies source code of cloud functions, which are small applications or application components, while the service provider is responsible for provisioning the infrastructure, scaling and exposing a REST style API. This environment seems to be adequate for running scientific workflows, which in recent years, have become an established paradigm for implementing and preserving complex scientific processes. In this paper, we present work done on adaptation of a scheduling algorithm to FaaS infrastructure. The result of this work is a static heuristic capable of planning workflow execution based on defined function pricing, deadline and budget. The SDBCS algorithm is designed to determine the quality of assignment of particular task to specific function configuration. Each task is analyzed for execution time and cost characteristics, while keeping track of parameters of complete workflow execution. The algorithm is validated through means of experiment with a set of synthetic workflows and a real life infrastructure case study performed on AWS Lambda. The results confirm the utility of the algorithm and lead us to propose areas of further study, which include more detailed analysis of infrastructure features affecting scheduling.

Keywords: Serverless · Cloud Functions · Workflow scheduling · Infrastructure testing.

1 Introduction

Scientific workflows are an established paradigm of implementing and preserving a scientific process. Workflows allow for modeling complex scientific procedures with help of abstractions over infrastructure or implementation details. Workflows are usually represented by an Directed Acyclic Graph (DAG) which enables to analyze them and determine the relations and dependencies between individual tasks. This allows for parallelization and execution planning.

In most cases, scientific workflows are executed by a Scientific Workflow Management System [5], which provides features required to execute the process. Additionally, management systems usually provide features aimed at automating and streamlining the process, like basic infrastructure management and some

fault tolerance. In order to execute the workflow we need two additional components, data to operate on and a computing infrastructure. The data is usually provided by the scientist or is an artifact produced by or directly included in the workflow. The infrastructure can be a personal computer, a HPC computing cluster or the cloud. Due to the features like: availability, pricing models and possibility to dynamically adapt to the workloads, cloud infrastructure seems to be a natural choice. One of the newest additions in cloud service provider's portfolios is the Function-as-a-Service. FaaS infrastructures provide computing power while taking the responsibility for on-demand provisioning of execution environments. Additionally FaaS offers an attractive pricing model where user is billed only for the actual time spent on computing, usually with 100 ms granularity. In case of such infrastructure, a developer is responsible only for supplying the application code and declaring memory requirements. Applications destined to run of FaaS are called *Serverless Applications* in order to emphasize the lack of operating on traditional servers or virtual machines, during the deployment and operation of the application. In contrast to Platform-as-a-Service a serverless application doesn't directly manage scaling and provides a limited run time for individual tasks. While those characteristics are a limitation, they allow the provider to supply a significantly greater scaling potential and speed of infrastructure provisioning.

Due to the unique features of FaaS we need to revisit some of the aspects of workflow execution and scheduling, as explained in [11]. One of such topics is the preparation of an execution plan. FaaS provides a highly elastic infrastructure, with unique performance characteristics, where CPU cycles are tied to the declared amount of memory and user is billed per 100ms of execution time. Furthermore functions don't have a persistent local storage, so each task needs to explicitly manage its inputs and outputs. This combination of features justifies the need for a dedicated scheduling algorithm. In this paper, we propose a Serverless Deadline-Budget Constrained Scheduling (SDBCS) algorithm, a heuristic which aims to prepare an execution plan satisfying budget and time constraints, while not introducing a high cost of plan preparation. SDBCS was implemented with help of HyperFlow [3], a proven and extensible workflow execution engine, written in JavaScript.

This paper is structured as follows. Sec. 2 elaborates on current body of knowledge related to scheduling workflow applications in FaaS infrastructures. Described references include analysis of the infrastructure, applications and possible scheduling algorithms. Sec. 3 describes in detail the used procedure of scheduling a workflow for FaaS. The environment, tooling and solution architecture is presented. The proposed scheduling algorithm is shown in Sec. 4. Scheduling problem is formally stated and methods for obtaining a plan are described in detail. Sec. 5 contains experiment results based on synthetic test package and a real life experiment involving usage of AWS Lambda functions. The paper concludes with Sec. 6 which give a summary of the paper and provide outlook for future work.

2 Related work

FaaS was originally designed to host event-based asynchronous applications, coming from Web or mobile usage scenarios. However, there is an ongoing work on finding other alternative use cases for FaaS, as shown in [2], which include event processing, API composition and various flow control schemes.

There are efforts which aim to implement frameworks, like pywren [7], which allow performing general purpose computing on FaaS clouds. One of the main features would be to enable dynamic transformation of applications to FaaS model while simultaneously providing deployment services, which would allow for seamless migration to cloud functions. The result would be a workflow application consisting of tasks which represent parts of the original application.

FaaS infrastructures, as a novelty, are subject to rapid changes. Work done in [10] describes the current details of FaaS provider offerings, service types, limitations and costs. The performance of cloud functions was further studied in [6] and [13]. Included results allow to construct the model the available performance and infrastructure provisioning characteristics like efficiency and limits.

In our earlier work [12] we proposed means to adapt scientific workflows to FaaS, using HyperFlow[1]. In [9] we proposed and validated a FaaS specific scheduling algorithm, which is used as a reference point for validating algorithm presented in this paper.

There is a plethora of workflow scheduling algorithms available for clouds based on virtual machines. Those algorithms can be adapted to FaaS, which would allow to benefit from the available body of knowledge. We chose the Deadline-Budget Constrained Scheduling algorithm [1] as a suitable for adaptation, due to its low complexity and good performance.

Workflow applications are a well studied field. In the case of this paper we evaluated the proposed algorithm with the help of available workflow test data set for Pegasus system, which is described in more detail in [8].

3 Serverless workflow execution

3.1 Scheduling the workflow

As presented in [9], executing workflows on FaaS is significantly different from execution on virtual machines. From the application's point of view, we need to distribute individual tasks across functions, so that the whole process can be executed with the imposed per task time limit. While the cloud provider is responsible for provisioning of the infrastructure, we need to declare suitable function configurations, so that the deadline and budget requirements are met. In case of the proposed algorithm, the output of the planning process is the assignment of tasks to function configurations. Each configuration is characterized by an amount of memory, which is proportional to available computing cycles per second, which in turn determines the execution time of tasks. If the cost of

[1] HyperFlow repository: https://github.com/hyperflow-wms/hyperflow

running the application is lower than the budget, and the makespan is shorter than the deadline, the scheduling is considered successful.

3.2 The environment, tools and solution architecture

In the course of our studies of FaaS infrastructures, we tested and evaluated multiple FaaS providers [6]. For the scope of this work we chose to work on Amazon infrastructure. We used AWS Lambda for running cloud functions and AWS S3 for cloud storage. The tight integration of both services, namely support for credential delegation greatly simplifies the deployment process, as cloud functions can hold delegated credentials required to access storage. At the time of writing this paper AWS Lambda imposes several limits on cloud functions. Functions are limited by time to 900 seconds, the amount of declared and used memory must be in the range of 128MB to 3008MB, which translates to available computing performance. Local storage available within a function environment is limited to 512MB, and deployment package (function code and auxiliary applications) need to fit in a 250MB package. Concurrent function executions are limited to 1000 instances. Table 1 includes function configurations used during the evaluation of the proposed algorithm. For the sake of simplicity of the application model some features of FaaS, like cold starts, are not directly addressed.

Table 1. Function configurations and prices. Note that memory size affects available CPU time (computing performance).

Memory size	Cost per 100ms of execution
256MB	$0.000000417
512MB	$0.000000834
1024MB	$0.000001667
1536MB	$0.000002501
2048MB	$0.000003334
2560MB	$0.000004168
3008MB	$0.000004897

HyperFlow served as a workflow execution engine. Workflows are represented as JSON structures containing a DAG. HyperFlow keeps track of the state of the application and is responsible for transforming tasks to function calls. Function calls are implemented as simple REST Calls, where REST APIs are exposed by the FaaS deployment. The deployment consists of prepared functions (part of HyperFlow package) which handle incoming calls and execute bundled components of the application.

The scheduler was implemented as a set of components, which include tools used to perform application test runs to gather performance characteristics of the tasks and the main scheduler module. The scheduler parses performance data, pricing, user defined constrains: deadline and budget, the output is an execution plan in form of a decorated workflow. The output workflow is a HyperFlow

compatible DAG, which includes mapping of each task to a target function type. Scheduler module is open source and its repository is publicly available[2].

Figure 1 presents the architecture used during the development and evaluation of the proposed algorithm. The whole process can be described as follows. Workflow application, in form of a JSON DAG, is supplied to the scheduler, which is responsible for producing an execution plan. The plan, a decorated DAG, is supplied to HyperFlow, which executes the application according to plan. Workflow manipulation, planning and execution management are performed outside of FaaS, on a dedicated machine. The execution of application's tasks is performed by calling a function with proper arguments. Each task is executed inside of individual function instance. Tasks share a common storage, which is available remotely, through a S3 protocol.

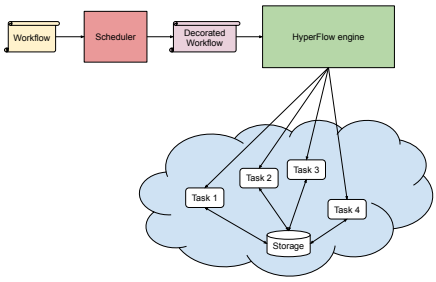

Fig. 1. Deployment diagram of workflow execution system

4 The scheduling algorithm

As mentioned in Sec. 1, there is a variety of available scheduling algorithms for workflow applications. We chose the Deadline-Budget Constrained Scheduling algorithm [1] as suitable for adapting to serverless infrastructures. The algorithm is a list scheduling algorithm applicable to cloud environments and operates on heterogeneous resources. The adaptation consisted of reimplementing the algorithm with the notion of functions instead of virtual machine oriented processors. This removed the need for part of algorithm responsible for selecting an available processor. Required functions are supplied by the FaaS provider in an on demand manner. Additionally calculation of storage and transfer costs were removed, as those basic functions are supplied and not directly contributing to costs.

[2] https://github.com/PawelBanach/CloudFunctionOptimizer

4.1 Serverless Deadline-Budget Constrained Scheduling algorithm (SDBCS)

The problem of scheduling can be defined as assigning individual tasks of workflow application to resources available from a heterogeneous environment. In this case resources are represented by cloud functions, available function configurations are listed in Table 1, where the number of functions in limited only by concurrency constraint. User supplies the application in form of a Directed Acyclic Graph (DAG), which can be represented by a tuple $G = \langle T, E, Data \rangle$ where $T = t_1, t_2, ..., t_n$ represents a set of tasks of workflow, and E represents edges connecting tasks, which model dependencies and control flow between tasks. Data represents input and intermediate information needed to run the application. Scheduling problem becomes a matter of finding a map $G : T \rightarrow F$, where F denotes the mentioned set of functions. The algorithm is a single step heuristic which meets the budget constrain and may or may not achieve the required deadline. Successful scheduling is achieved when all constraints are meet. The core functionality of the algorithm is based on sorting tasks according to upward rank, calculating sub-deadlines and quality score for each task on each resource. Sub deadline is inferred from user supplied makespan deadline. Quality is calculated based on task execution time on a given resource. Description of the algorithm uses notation presented in Table 2.

Table 2. Symbols and notation used for algorithm description.

Symbol	Description
t_{curr}	currently scheduled task
$rank_u(t)$	rank of task t
$ET(t)$	execution time of task t
$succ(t)$	successors of task t
$FT(t, r)$	finish time of task t on resource r
$FT_{min}(t, r)$	minimum finish time of task t on resource r
$FT_{max}(t, r)$	maximum finish time of task t on resource r
$Cost(t, r)$	cost of executing task t on resource r
$Cost_{min}(t)$	minimum cost of executing task t
$Cost_{max}(t)$	maximum cost of executing task t
$Cost_{best}(t)$	cost of fastest execution of task t
$Cost_{cheapest}$	minimum cost of executing all tasks
$AC(t)$	assigned cost of running task t
Δ_{Cost}	spare budget

Specific elements of the algorithm operate base on the following rules. The spare budget is calculated with the formula:

$$\Delta_{Cost} = \Delta_{Cost} - [AC(t_{curr}) - Cost_{min}(t_{curr})] \tag{1}$$

where the spare budget is the difference between available budget and the cheapest assignment for unscheduled tasks. The initial spare budget is expressed as:

$$\Delta_{Cost} = BUDGET_{user} - Cost_{cheapest} \tag{2}$$

and

$$Cost_{cheapest} = \sum_{t_i \in T} Cost_{min}(t_i) \tag{3}$$

Task selection and priority is based on a computed rank of tasks. The rank represents the length of longest path from task to the exit node with addition of average execution time of task:

$$rank_u(t_i) = \overline{ET}(t_i) + max_{t_{child} \in succ(t_i)}\{rank_u(t_{child})\} \tag{4}$$

Where $\overline{ET}(t_i)$ represents the average execution time of task over available resources and the latter part of equation represents the maximum of ranks of all immediate successors of task t_i.

The budget available for task execution is expressed as:

$$CL(t_{curr}) = Cost_{min}(t_{curr}) + \Delta_{Cost} \tag{5}$$

which represents the minimum execution cost with addition of spare budget.

The sub-deadline is defined for each task as:

$$DL(t_{curr}) = min_{t_{child} \in succ(t_{curr})}[DL(t_{child}) - ET_{min}(t_{child})] \tag{6}$$

where tasks' individual deadline is calculated as the minimum of difference between subsequent tasks' deadline and minimum execution time of current task.

The $Time_Q$ and $Cost_Q$ represent time and cost quality of assigning task to resource, quality measures are expressed as:

$$Time_Q(t_{curr}, r) = \frac{\Omega * DL(t_{curr}) - FT(t_{curr}, r)}{FT_{max}(t_{curr}) - FT_{min}(t_{curr})} \tag{7}$$

$$Cost_Q(t_{curr}, r) = \frac{Cost_{best}(t_{curr}) - Cost(t_{curr}, r)}{Cost_{max}(t_{cur}) - Cost_{min}(t_{cur})} * \Omega \tag{8}$$

and:

$$\Omega = \begin{cases} 1 \text{ if } FT(t_{curr}, r) < DL(t_{curr}) \\ 0 \text{ otherwise} \end{cases} \tag{9}$$

Time and Cost qualities aim to represent the distance of studied solution in the range between *best* and *worst* case scenarios. In case of time, the range of values spans between the sub-deadline and minimum execution time, while in case of cost boundaries are set at the minimum and maximum execution costs.

The Ω parameter is responsible for complying with the deadline set for current task.

The final quality measure is expressed as:

$$Q(t_{curr}, r) = Time_Q(t_{curr}, r) + Cost_Q(t_{curr}, r) * \frac{Cost_{cheapest}}{Budget_{Unconsumed}} \quad (10)$$

the equation combines both quality measures with addition of weighting the $Cost_Q$ with the ratio of cheapest execution cost to unconsumed budget.

SDBCS is presented as Algorithm 1. The process included in algorithm can be described as follows. Initialization of the algorithm requires user to supply a workflow graph, a deadline and available budget. If the budget is less than the lowest possible cost of executing the workflow the algorithm indicates it is impossible to create an appropriate plan. Next step (lines 3-4) determines if the supplied budget is more than the highest possible cost of execution, in that case each task is scheduled to execute on the fastest resource. Line 6. is responsible for assigning the initial value of spare budget, next step is to calculate ranks and task priority. Lines 8-15 contain the main scheduling loop, which iterates over task in order of priority. Quality measure is computed for currently scheduled task against all available resources, based on that the best resource is selected. The final part of the loop is the update of spare budget which is calculated according to equation 1.

Algorithm 1 Serverless Deadline-Budget Constrained Scheduling algorithm

Require: DAG, time (D_{user}), budget (B_{user})
1: **if** $B_{user} < Cost_{min}(DAG)$ **then**
2: **return** no possible schedule
3: **else if** $B_{user} > Cost_{max}(DAG)$ **then**
4: **return** schedule map on the most expensive resource
5: **end if**
6: $\Delta_{Cost} \Leftarrow BUDGET_{user} - Cost_{cheapest}$
7: Compute upward rank ($rank_u$) and sub-deadline for each task
8: **while** there is unscheduled task **do**
9: $t_{curr} \Leftarrow t$ next ready task with highest rank
10: **for** $r \in$ resources **do**
11: Calculate quality measure $Q(t_{curr}, r)$
12: **end for**
13: $r_{selected} \Leftarrow r$ with highest quality value
14: assign t_{curr} to $r_{selected}$
15: $\Delta_{Cost} \Leftarrow \Delta_{Cost} - [AC(t_{curr}) - Cost_{min}(t_{curr})]$
16: **end while**
17: **return** schedule map

5 Evaluation and results

5.1 Scheduling performance

The proposed algorithm was evaluated in a series of experiments. Experiments are meant to test the scheduling success rate for a set of test workflow applications in multiple input parameters. Tests were designed with the use of Montage, which in recent years became an established benchmark application for workflow scheduling systems. Montage is a astronomical mosaic application, which combines an array of smaller images into larger mosaics. The application is composed of several steps and contains many tasks executed in parallel, thus it is suitable to validate scheduling performance. Testing workflows were obtained from the workflow repository available at Pegasus system homepage[3], described in more detail in [4] and [8]. It is important to note that Montage was chosen as a utility to verify algorithm's performance and not as an ideal application to run in FaaS environment. Test package contains 220 workflows, with task counts ranging from 50 to 1000. Workflows were converted to HyperFlow format and task run time for each resource was estimated. The estimation was made based on package-supplied synthetic run time and function performance metrics from our earlier work [6]. Synthetic run time was treated as time taken by task execution on slowest cloud function. Other run times, for faster function configurations, were obtained by simply scaling it by expected function performance. In a real world use case, one would be required to supply run time of each task on each function configuration.

The experiments were conducted for Serverless Deadline-Budget Constrained Scheduling (SDBCS) and Serverless Deadline-Budget Workflow Scheduling (SDBWS) algorithm. SDBWS is described in more detail in [9]. The main difference of SDBWS is that it operates on tasks grouped in levels, which are assigned a global sub-deadline, whereas SDBCS treats each task as a separate entity. Additionally SDBWS utilizes different formulas to calculate quality. SDBCS can be treated as a more general derivative of SDBWS and is expected to provide better performance. Due to the focus of this paper, experiments were narrowed to test only two mentioned algorithms.

The set of deadline and budget parameters were generated based on minimal and maximal possible values. Specific values at $0.3, 0.5$ and 0.7 points of range were chosen. The final values of deadline and budget were calculated for each workflow with the following equations:

$$Deadline_{user} = Deadline_{min} + a_D * (Deadline_{max} - Deadline_{min}) \quad (11)$$

$$Budget_{user} = Budget_{min} + a_B * (Budget_{max} - Budget_{min}) \quad (12)$$

The results of scheduling experiments are presented in Figures 2, 3 and 4. Each figure contains results for a specified value of budget parameter, whereas

[3] https://download.pegasus.isi.edu/misc/SyntheticWorkflows.tar.gz

the X axis spans across multiple values of a_D parameter. Results show that SDBCS overall performance is better than SDBWS, with exception of smaller value of deadline, where both algorithms presented low success rate. In case of $a_D = 0.5$ SDBCS clearly delivers better performance and for $a_D = 0.7$ SDBCS advantage over SDBWS is present but not as significant. The case of $a_D = 0.7$ and $a_B = 0.7$ results in both algorithms succeeding at scheduling all test workflows.

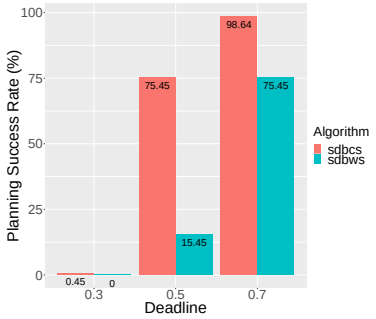

Fig. 2. Scheduling success rate with budget $a_B = 0.3$ and deadline $a_D \in \{0.3, 0.5, 07\}$

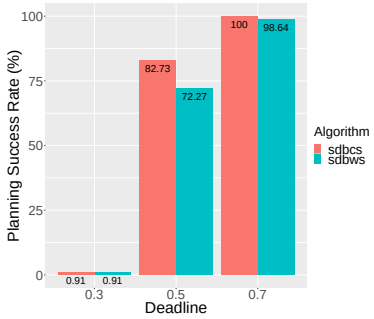

Fig. 3. Scheduling success rate with budget $a_B = 0.5$ and deadline $a_D \in \{0.3, 0.5, 07\}$

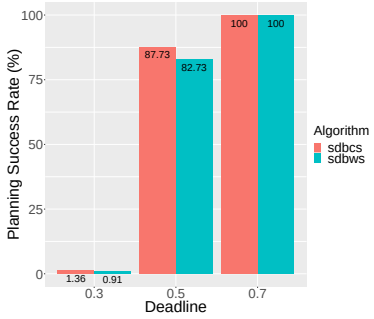

Fig. 4. Scheduling success rate with budget $a_B = 0.7$ and deadline $a_D \in \{0.3, 0.5, 07\}$

5.2 Tests on physical infrastructure

As part of validation of the proposed algorithm, we performed a real life experiment. The test used physical infrastructure in order to prove the applicability

of the solution. Procedure included scheduling a sample workflow, namely Montage application containing 43 tasks. The next step was to run the application on setup described in Sec. 3.2, where tasks were executed on AWS Lambda. Figure 5 contains a Gantt chart depicting the *trace* of execution, X axis represents time, each bar represents run time of a single task. Task types are distinguished by color, distinction between types allows to determine dependencies between tasks. Transparent bars represent planned execution while opaque are executions measured in real life. The chart allows to visually inspect the accuracy of planning. In the presented case, plan closely matched the real life execution, and only with 4 tasks, the execution was started slightly after the planed time.

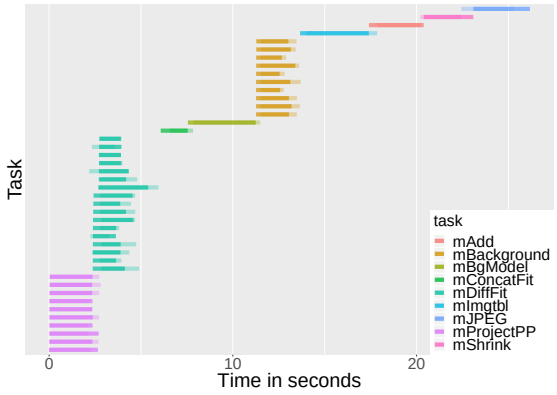

Fig. 5. Gantt chart depicting a trace of Montage execution. Execution of each task is represented by a opaque bar, while a transparent bar represents scheduled execution.

6 Conclusions and future work

Presented adaptation of scheduling algorithm was made after careful analysis of target infrastructure and provided insight into characteristics of running workflows on FaaS infrastructures. Obtained results confirm, that the presented Serverless Deadline-Budget Constrained Scheduling algorithm is capable of producing valid execution plans according to supplied parameters. Experiments with scheduling a Montage workflow proven that SDBCS achieves better results than the previously studied SDBWS algorithm. Real life infrastructure tests also illustrate, that the generated execution plan is valid in practical applications.

Future work includes further study of workflow scheduling algorithms and exploring new methods of adapting them to FaaS infrastructures. Additionally, our work on studying commercially available infrastructures, led us to conclusion that the behaviour of FaaS is still not completely explored. Functions tend to experience phenomena like execution throttling or delays, which have an impact on workflow execution and could be accounted for in scheduling algorithms.

Acknowledgements This work was supported by the National Science Centre, Poland, grant 2016/21/B/ST6/01497.

References

1. Arabnejad, H., Barbosa, J.G., Prodan, R.: Low-time complexity budget–deadline constrained workflow scheduling on heterogeneous resources. Future Generation Computer Systems **55**, 29–40 (2016)
2. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., et al.: Serverless computing: Current trends and open problems. In: Research Advances in Cloud Computing, pp. 1–20. Springer (2017)
3. Balis, B.: Hyperflow: A model of computation, programming approach and enactment engine for complex distributed workflows. Future Generation Computer Systems **55**, 147–162 (2016)
4. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H., Vahi, K.: Characterization of scientific workflows. In: 2008 third workshop on workflows in support of large-scale science. pp. 1–10. IEEE (2008)
5. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: An overview of workflow system features and capabilities. Future Generation Computer Systems **25**(5), 528–540 (2009)
6. Figiela, K., Gajek, A., Zima, A., Obrok, B., Malawski, M.: Performance evaluation of heterogeneous cloud functions. Concurrency and Computation Practice Experience **accepted** (2017)
7. Jonas, E., Pu, Q., Venkataraman, S., Stoica, I., Recht, B.: Occupy the cloud: Distributed computing for the 99%. In: Proceedings of the 2017 Symposium on Cloud Computing. pp. 445–451. ACM (2017)
8. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Future Generation Computer Systems **29**(3), 682–692 (2013)
9. Kijak, J., Martyna, P., Pawlik, M., Balis, B., Malawski, M.: Challenges for scheduling scientific workflows on cloud functions. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). pp. 460–467. IEEE (2018)
10. Lee, H., Satyam, K., Fox, G.C.: Evaluation of production serverless computing environments. In: Proceedings of the 3nd International Workshop on Serverless Computing. ACM ((In Print))
11. Malawski, M.: Towards serverless execution of scientific workflows-hyperflow case study. In: WORKS@ SC. pp. 25–33 (2016)
12. Malawski, M., Gajek, A., Zima, A., Balis, B., Figiela, K.: Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions. Future Generation Computer Systems **(In Print)** (nov 2017). https://doi.org/10.1016/j.future.2017.10.029, http://linkinghub.elsevier.com/retrieve/pii/S0167739X1730047X
13. Pawlik, M., Figiela, K., Malawski, M.: Performance evaluation of parallel cloud functions. Poster presented at ICPP 2018 : International Conference on Parallel Processing, Eugene, Oregon, USA (2018)