Challenges for Scheduling Scientific Workflows on Cloud Functions

Joanna Kijak, Piotr Martyna, Maciej Pawlik, Bartosz Balis, Maciej Malawski
AGH University of Science and Technology
Department of Computer Science
Krakow, Poland
Email: malawski@agh.edu.pl

***Abstract*—Serverless computing, also known as Function-as-a-Service (FaaS) or Cloud Functions, is a new method of running distributed applications by executing functions on the infrastructure of cloud providers. Although it frees the developers from managing servers, there are still decisions to be made regarding selection of function configurations based on the desired performance and cost. The billing model of this approach considers time of execution, measured in 100ms units, as well as the size of the memory allocated per function. In this paper, we look into the problem of scheduling scientific workflows, which are applications consisting of multiple tasks connected into a dependency graph. We discuss challenges related to workflow scheduling and propose the Serverless Deadline-Budget Workflow Scheduling (SDBWS) algorithm adapted to serverless platforms. We present preliminary experiments with a small-scale Montage workflow run on the AWS Lambda infrastructure.**

***Index Terms*—FaaS, serverless computing, cloud functions, scientific workflow, task scheduling**

1. Introduction

Function as a Service, or serverless computing, is a relatively new approach to distributed computing in the cloud, wherein small units of code (cloud functions) can be deployed and run on demand while server management and dynamic scaling are managed by the cloud infrastructure. Moreover, the current implementations of serverless platforms, such as AWS Lambda, Google Cloud Functions, Azure Functions or IBM Cloud Functions, allow executing custom binary code on their infrastructures, making them usable not only for specific Web application frameworks, but for general purpose computing tasks that can fit into this model. Examples include scientific workflows, which are applications consisting of multiple tasks connected into a dependency graph. Many of scientific workflows include a large number of parallel fine-grained tasks which can naturally fit into the cloud functions.

Serverless computing is an attractive execution model for scientific workflows, as the provider takes care of such resource management issues as resource provisioning and autoscaling. Nevertheless, there are still challenges related to scheduling of scientific workflows on cloud functions. For example, functions are heterogeneous in terms of the memory and computing power assigned to them. The cost is based on the resource consumption and it is charged per every 100ms of the CPU time used. Also, there is a strict limit on the function's execution time, e.g. 300 seconds on AWS Lambda. Moreover, when scheduling workflows, we need to take into account the various overheads the cloud providers may introduce, parallelism limits, and data access model. All these challenges motivate us to rethink the scheduling problem and investigate the approaches suitable for the serverless model.

In this paper, we focus on the challenges for scheduling scientific workflows on the example of the Serverless Deadline-Budget Workflow Scheduling (SDBWS), a heuristic adapted to run all tasks in the FaaS execution model. We present a preliminary experimental evaluation of this heuristic using our HyperFlow workflow engine and small-scale workflows run on the AWS Lambda infrastructure.

The paper is structured as follows: in Section 2 we give a short overview of the recent developments in using serverless platforms for scientific workflows and relevant scheduling approaches. Next, in Section 3 we discuss in detail the challenges related to scheduling workflows in serverless platforms. Section 4 presents the scheduling algorithm we developed, which is a list scheduling heuristic adapted to serverless environment. In Section 5 we present the preliminary results, and in Section 6 we discuss the conclusions and future work.

2. Related Work

In our earlier work we have done preliminary experiments with scientific workflows [1] on AWS Lambda and Google Cloud Functions using HyperFlow, a lightweight workflow engine. A more advanced solution was presented in [2], where the implementation of a hybrid model combining FaaS with IaaS was evaluated, showing the benefits of such a hybrid approach.

In [3], we presented a performance evaluation of cloud functions by running benchmarks on infrastructures of major cloud function providers: AWS Lambda, Azure Functions, Google Cloud Functions and IBM OpenWhisk.

While serverless computing is still a relatively new paradigm, much work has been done regarding the potential applications of the new infrastructure. Examples include running containers on serverless infrastructures [4] and low-latency, massively parallel, video processing using cloud functions [5].

There are multiple static algorithms for cloud workflow scheduling. One example is Deadline-Budget Workflow Scheduling (DBWS) [6], which we selected as the one that can be easily adapted to the FaaS model. An example of dynamic scheduling is given in [7], which describes a bag-of-tasks scheduler with budget constraints.

In this paper, we study the application of the FaaS infrastructure from the perspective of the end user. Consequently, we are not concerned about the implementation details of the particular infrastructure. However, there is ongoing research on scheduling on the provider side, which may also have impact on workflow execution. Some examples are described in [8], which studies the aspects of FaaS implementation like environment reusability and infrastructure provisioning overhead.

3. Workflow Execution in Serverless Model

Serverless workflow execution differs from execution in IaaS clouds in terms of architecture and scheduling. Let us discuss these issues in this section.

3.1. Workflow Execution Architecture

There are several possible workflow execution architectures leveraging the serverless infrastructure, discussed in our previous article [1]. For the purpose of this paper we have implemented the direct execution model in the HyperFlow workflow engine [9]. The model of computation implemented by HyperFlow belongs to the *process networks* family. In HyperFlow, workflows are multi-graphs, described in a simple JSON format, where nodes are workflow activities (called *processes*), while edges represent data and control flow between them. While the workflow description language is simple, it is highly expressive, allowing implementation of diverse complex workflow patterns [9]. The engine maintains the workflow execution state, and whenever a workflow process is activated, it calls a user-defined JavaScript function which usually is the entry point to invoking the actual scientific procedure associated with the given workflow activity. This is a flexible execution model that gives advanced users low-level control over how workflow activities are implemented. When using distributed computing infrastructures, the entry point functions use their APIs to orchestrate execution of workflow jobs on distributed resources. We have previously shown that this model can be successfully applied to both IaaS and PaaS clouds [10]. Here, we extend it to serverless infrastructures.

The serverless workflow execution architecture is depicted in Fig. 1. The workflow engine is deployed outside the serverless infrastructure, e.g. on the user's laptop, or a dedicated virtual machine in a cloud, and invokes the cloud functions directly (via a cloud API, or an API gateway). For task scheduling, we implement an approach in which the scheduler prepares the plan (mapping between tasks and resources) prior to workflow execution. The plan is then passed to the engine in the form of a decorated workflow description.

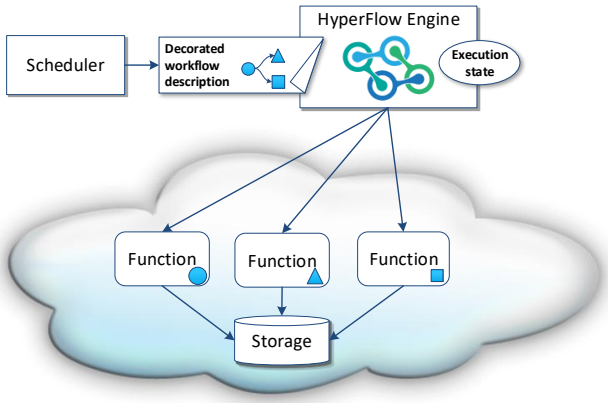


Figure 1: Workflow Execution Architecture

3.2. Workflow Scheduling Challenges

From the perspective of the user, the FaaS model leads to the following challenges regarding workflow scheduling:

1) Which size of cloud functions should be allocated to each task of a workflow?
2) Which tasks should be executed on FaaS and which ones on IaaS?
3) What is the performance variability of the cloud functions infrastructure and how to deal with it?
4) What are the limits of concurrency that we can expect when running multiple tasks as cloud functions in parallel?
5) How to address the problem of data transfer between tasks?

Challenge (1) is a fundamental problem of scheduling workflows on heterogeneous resources and it depends on the optimization criteria and constraints, most importantly time and cost. In the basic variant we can assume idealized infrastructure in which all functions submitted for execution start immediately and there is no startup delay. Various scheduling heuristics can be adapted to this model to produce plans to be used by the workflow engine.

We can distinguish two approaches: static and dynamic scheduling. In the static approach, workflow tasks are allocated to computing resources before the execution, e.g. based on data from previous executions. In the dynamic approach, the execution is monitored and the schedule can be adjusted at runtime, e.g. tasks can be assigned to faster functions if the deadline is at risk.

Challenge (2) is related to the specific constraint FaaS providers impose on the execution time of cloud functions,

for example 300 or 540 seconds. Obviously, tasks that may run longer than the limit need to be executed on standard VMs, so a hybrid execution model is necessary [2]. In that case there still remains the problem of VM provisioning, i.e. choosing the VM size and deciding when to start and terminate such VMs. In this paper, we do not solve this problem, since our SDBWS algorithm works only for workflows which can be run entirely in FaaS.

Challenges (3) and (4) stem from the difference between the real infrastructure and its idealized model. Due to performance variability of the cloud, the actual execution will diverge from its schedule obtained by the scheduling heuristic, and the constraints such as deadline will be violated. This effect will be even more harmful when the concurrency limit is reached, since the tasks which cannot execute in parallel may delay the workflow considerably.

To address challenges (3) and (4) we can use performance studies of FaaS providers. In our earlier work we started a systematic monitoring of cloud functions to gain insights into the performance problems [3]. The data gathered in these studies can provide input for the scheduling algorithms, but research in this area is still needed [11].

Finally, challenge (5) is solved by the infrastructure model: the cloud functions are in general stateless, so there is a need to use some external storage for intermediate data. Cloud object storage can thus be used, as in [1], [2]. This differs from traditional scheduling on heterogeneous resources, where data is transferred directly between resources.

In this paper, we focus on challenge (1) by adapting an existing DBWS heuristic to the serverless workflow execution model, and via experiments on real clouds we observe how challenges (3) and (4) influence the overall performance.

4. Static Scheduling Solution

The main purpose of scheduling is to produce an execution plan which allows completing all tasks of a workflow within given time and cost constraints.

As described in section 2, there is a variety of scheduling algorithms applicable for cloud environments. For the needs of this study, we have chosen the Deadline-Budget Workflow Scheduling algorithm [6], with some modifications adapting it to fit the serverless computing model. This particular algorithm was chosen because it is a list scheduling algorithm, which utilizes a heuristic approach based on the efficient PEFT algorithm [12] and is well suited for heterogeneous cloud environments by incorporating a billing model. Moreover, it was well documented and easy to adapt. In its original form, the estimation of minimum and maximum cost of schedules is performed using PEFT, while in the case of FaaS this step is not needed.

4.1. Formal Definition of the Problem

Scientific workflow can be represented as a Directed Acyclic Graph (DAG), which can be defined as $G = \langle T, E, Data \rangle$ where $T = \{t_1, t_2, ..., t_n\}$ is a set of tasks, while E represents the set of edges denoting task data dependencies. A dependency guarantees that a child task cannot be executed before all parent tasks have finished and transferred required data. $Data$ denotes the files that need to be available in order to execute a given task. Tasks in a DAG can be grouped into levels based on the path length from the entry node to a task's node.

Cloud computing platforms offer many heterogeneous function types that have different cost and performance. F denotes the set of function types that are available for given provider. Different infrastructure providers tend to offer their own unique set of function types.

The scheduling problem can be defined as finding a map function $g : T \rightarrow F$ between task and function types, in order to meet the requirements defined for a workflow. The sought solution should comply with time and cost constraints given by the user, whenever possible.

4.2. Proposed Approach

Our solution is based on the HyperFlow engine and includes the following steps:

1) decorate the workflow description file with additional information about where to execute the functions,
2) deploy the function executor on FaaS infrastructure with specified memory constrains,
3) for workflow execution use HyperFlow extended with a command responsible for invoking the function according to information from decorated DAG.

This approach is based on the information available at the moment when the execution process begins. One type of information is the execution time of each function, gathered from previous runs. This allows for making assumptions about future runtimes. Another type of data is the time of function execution depending on the function configuration parameters. The most interesting property is memory size, which directly relates to available computing power and cost. The greater is the memory size, the more computing power is available.

4.3. Serverless Deadline-Budget Workflow Scheduling Algorithm

Deadline-Budget Workflow Scheduling [6] is a heuristic strategy that allows obtaining a schedule in a single step. The strategy always meets the deadline constraint and tries to accomplish the budget constraint. The algorithm has two phases: task selection and resource selection.

The main idea behind DBWS is to sort tasks by an upward rank, then calculate sub-deadlines, and finally to calculate a quality score for each task on each resource. To calculate the upward rank, the average execution time of a task on all resources is used. The sub-deadline is assigned to each task based on the overall user-provided deadline,

proportionally to the maximum execution time of each level in the graph. The final quality score is calculated for each task on each resource based on the sub-deadline information and constraints provided by the user. If the finish time of the task on a resource takes longer than its sub-deadline, then the score on that resource is lower. This guarantees that the schedule plan meets deadline expectations. The resource with the highest score is chosen as the designated resource. This approach has been developed for virtual machines and the pay-per-hour billing model. However, by making some modifications it can be adapted to the function model.

One of the features which distinguish FaaS from IaaS is the granularity of the accounting. In the case of IaaS it was possible to exploit the hourly billing rate by running additional tasks on machines which finished their workload early. In the case of FaaS this optimization is not possible, due to millisecond grained accounting.

When a task on a Virtual Machine completes in a time shorter than one hour, the next tasks scheduled on the same machine can be counted with zero cost. Each task executed as a function is treated independently and the cost is always greater than zero.

We propose a serverless version of the Deadline-Budget Workflow Scheduling (SDBWS) algorithm using the notation based on [6]:

- t_{cur} denotes the current task to be scheduled,
- $Cost(task, r)$ denotes cost for a given task on the resource of type r,
- $ST(task, r)$ denotes start time for a task on the resource of type r,
- $FT(task, r)$ denotes finish time for a task on the resource of type r,
- $Cost_{low}(DAG)$ and $Cost_{high}(DAG)$ represent total execution cost for scheduling each task on the resource with the lowest and highest cost among all possible resources available,
- $Cost_{min}(task)$ and $Cost_{max}(task)$ denote minimum and maximum time for task among all tested resources,
- $FT_{min}(task)$ and $FT_{max}(task)$ denote minimum and maximum finish time for a task among all tested resources,
- $l(t_i)$ denotes level of task t_i. It is an integer value representing the length of the longest path from the entry node to t_i

$$l(t_i) = 1 + \max_{t_p \epsilon predecessors(t_i)} l(t_p),$$

  where $l(t_{entry}) = 1$,
- $DAG_{makespan}$ is equal to $FT(t_{exit}) - ST(t_{entry})$.

4.4. Resource Selection

To select the best resource for a given task, we evaluate the relation between cost and time constraints. We define a sub-deadline for each task, based on the deadline of the whole workflow and on levels of the graph. Then, we define maximum execution time for each level:

$$Level^j_{execution} = \max_{l(t_i)==j} \{ET_{max}(t_i)\}$$

where ET_{max} represents the maximum execution time for task t_i among all resources. Next, we distribute the user deadline (D_{user}) between all levels. The sub-deadline value is computed recursively by going through the graph starting from the first level.

$$Level^j_{DL} = Level^{j-1}_{DL} + D_{user} * \frac{Level^j_{execution}}{\sum_{1 \le j' \le l(t_{exit})} Level^{j'}_{execution}}$$

All tasks belonging to the same level have same sub-deadline.

$$S_{DL}(t_{cur}) = \{Level^j_{DL} | l(t_i) == j\}$$

The resource is selected based on the combination of two factors: cost and time, trying to keep the best balance between them. We define time and cost quantities for a current task on every type of resource. These quantities are normalized by their maximum values

$$Time_Q(t_{cur}, r) = \frac{\xi * S_{DL}(t_{cur}) - FT(t_{cur}, r)}{FT_{max}(t_{cur}) - FT_{min}(t_{cur})}$$

$$Cost_Q(t_{cur}, r) = \frac{Cost_{max}(t_{cur}) - Cost(t_{cur}, r)}{Cost_{max}(t_{cur}) - Cost_{min}(t_{cur})} * \xi$$

where

$$\xi = \begin{cases} 1 & \text{if } FT(t_{cur}, r) < S_{DL}(t_{cur}) \\ 0 & \text{otherwise} \end{cases}$$

$Time_Q$ denotes how far the task's finish time on the resource r is from the level sub-deadline. $Cost_Q$ measures how much less the cost on resource r is than the cost on the most expensive resource. ξ is a flag that makes qualities irrelevant in the case if the task on resource r exceeds the sub-deadline constraint.

A resource is selected based on a quality measure which is computed as follows:

$$Q(t_{cur}, r) = Time_Q(t_{cur}, r) * (1 - C_F) + Cost_Q(t_{cur}, r) * C_F$$

where C_F, a cost-efficient factor is a tradeoff factor defined as:

$$C_F = \frac{Cost_{low}(DAG)}{B_{user}}$$

$Time_Q$ and $Cost_Q$ parameters represent user preferences as they are weighted by the ratio of the cheapest workflow execution. A lower value of C_F means that the user prefers to pay more and execute the whole workflow faster, while the higher value of C_F results in choosing cheaper resources.

During the test runs we measure start and end time of each task within the function, i.e. on the provider side. It is then used to calculate the execution time and cost of the task on the resource. As a result, the makespan which is calculated afterwards as $FT(t_{exit}) - ST(t_{entry})$ includes

the overheads of running the tasks: there is always a delay between invocation of a cloud function by the workflow engine and the start of task execution on the provider infrastructure. Inclusion of these overheads is crucial because they can have a big impact on the whole workflow execution.

Algorithm 1 Serverless Deadline-Budget Workflow Scheduling

Require: DAG, time (D_{user}) and budget (B_{user})
1: Sort all tasks based on their level
2: **if** $B_{user} < Cost_{low}(DAG)$ **then**
3: **return** no possible schedule
4: **else if** $B_{user} > Cost_{high}(DAG)$ **then**
5: **return** schedule map on the most expensive resource
6: **end if**
7: Compute the sub-deadline value for each task
8: **while** there is unscheduled task **do**
9: **for** $r \in$ resources **do**
10: Calculate quality measure $Q(t_{cur}, r)$
11: **end for**
12: $r_{selected} \Leftarrow r$ with highest quality measure
13: assign t_{cur} to $r_{selected}$
14: **end while**
15: **return** schedule map

SDBWS is shown in Algorithm 1. First, we sort tasks by their level. In the next steps, we check the possibility of finding a schedule under given constraints. If the cost computed as the sum of costs of running each task on the cheapest resource is lower than the given budget, there is no possibility of executing the workflow without exceeding the budget. If the budget is higher than the cost of running each task on the most expensive resource, tasks are scheduled to run on this resource. If neither of these conditions is satisfied, we compute a sub-deadline value for each task, using the formula presented earlier. A sub-deadline defines the maximum amount of time that can pass in order to achieve the deadline constraint. Next, the algorithm schedules every task to the best suited resource, as long as there are unscheduled tasks in the workflow. To do so, we select a task and calculate a quality measure for it on every resource. Then, we choose a resource with the highest quality measure and assign this resource to the task.

The main differences between SDBWS and DBWS can be summarized as follows:

- scheduling a workflow on a set of homogeneous resources by using PEFT algorithm is not needed since we use a separate resource for each task,
- the number of available resources is not limited,
- since we cannot reuse the resource as is done with VMs, we do not need to track which of the given resources are currently in use,
- the user is billed for each 100ms of executed code, unlike in the IaaS model,
- exchange of data between tasks is based on transfers from and to a cloud storage and the time needed for communication is included in the task's execution time,
- there is no requirement to sort the tasks by their upward rank,
- to take into account the overheads, the makespan used to calculate the sub-deadlines includes all the overheads measured during runs on homogeneous resources.

5. Preliminary Results

To evaluate the results we used AWS Lambda to execute the small-scale 0.25-degree Montage [13] workflow which consists of 43 tasks. It is an astronomical application often used for evaluation of workflow systems. First, to obtain estimates of task runtimes, we performed the test runs of the workflow on homogeneous resources, i.e. functions with 256, 512, 1024 and 1536 MB memory allocated, several times on each resource type. Having the data about each task average execution time, we set the deadline and budget constraints and performed scheduling using the SDBWS algorithm. Subsequently, we executed the workflow using the resources assigned to tasks, measuring each task execution time as well as the workflow makespan and cost.

In order to keep the constraints feasible, they are computed within the range given by boundary values. We calculate the total makespan of the workflow on the resources with highest and lowest associated cost as the minimum (min_D) and maximum (max_D) deadline value. The corresponding execution costs are minimum (min_B) and maximum (max_B) cost boundary values for the workflow. Having these values we define user's budget and deadline parameters described as:

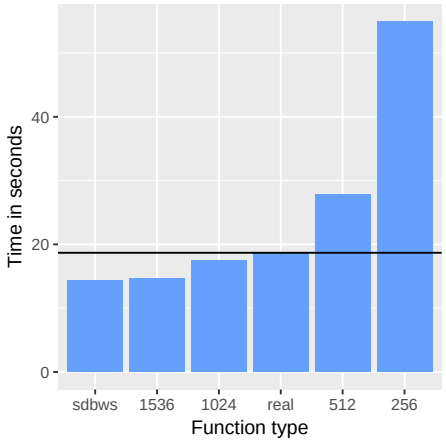
$$D_{user} = min_D + a_D * (max_D - min_D)$$

$$B_{user} = min_B + a_B * (max_B - min_B)$$

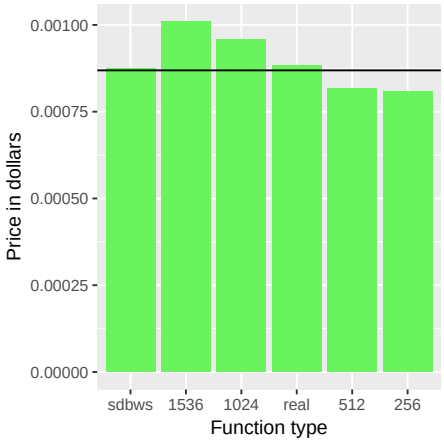
TABLE 1: Deadline and budget parameters used in experiments

a_D	a_B	Deadline	Budget
0.1	0.3	18.6 s	$0.00086
0.3	0.7	26.7 s	$0.00094
0.7	0.3	42.8 s	$0.00086

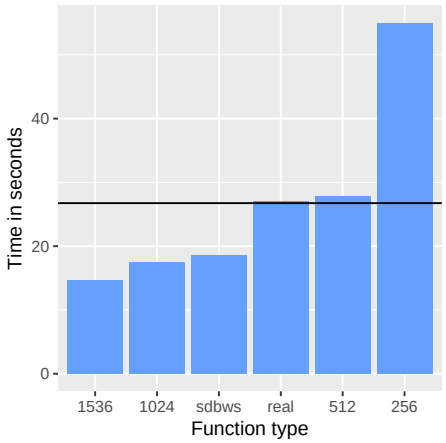
Deadline and budget parameters given by a_D and a_B can be selected from range [0...1]. We conducted the experiment with various deadline and budget parameters. In this paper, we present the results for the most interesting parameter values as shown in Table 1. For other values of budget and deadline parameters, which were similar to the cost and makespan in homogeneous executions, the results were correct but less interesting. The most of the selected resources were of a single type. These solutions confirm that the algorithm works properly, but we have decided to show here the solutions with more diverse resource assignments.



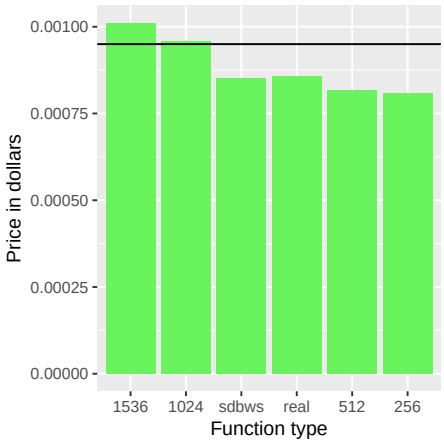
(a) Execution time $a_D = 0.1$ and $a_B = 0.3$



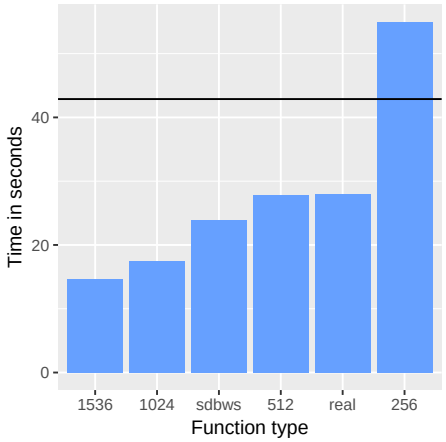
(b) Execution cost $a_D = 0.1$ and $a_B = 0.3$



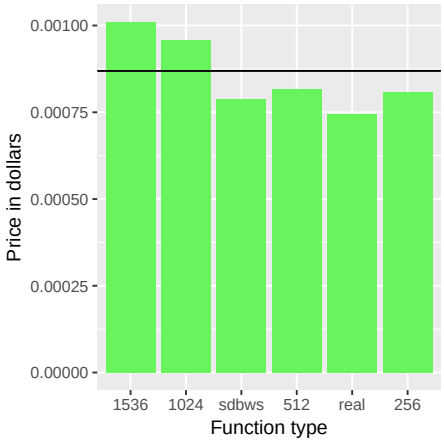
(c) Execution time $a_D = 0.3$ and $a_B = 0.7$



(d) Execution cost $a_D = 0.3$ and $a_B = 0.7$



(e) Execution time $a_D = 0.7$ and $a_B = 0.3$



(f) Execution cost $a_D = 0.7$ and $a_B = 0.3$

Figure 2: Comparison of makespans and costs of execution on homogeneous resources and scheduled by SDBWS

The execution results for the selected parameters are shown in Figure 2. The horizontal lines delineate the deadline and budget constraints, respectively. The bar described as *sdbws* represents the theoretical makespan and cost that the workflow would have if there were no delays in execution between the consecutive tasks. Obviously, as an ideal solution it is better in terms of both time and cost than any real execution. It shows how big the influence of the delays can be on the overall workflow makespan.

TABLE 2: Percentage comparison of the results with slower homogeneous resource

a_D	a_B	Faster	More expensive
0.1	0.3	33.3%	7.9%
0.3	0.7	3.07%	4.9%
0.7	0.3	49.0%	-7.8%

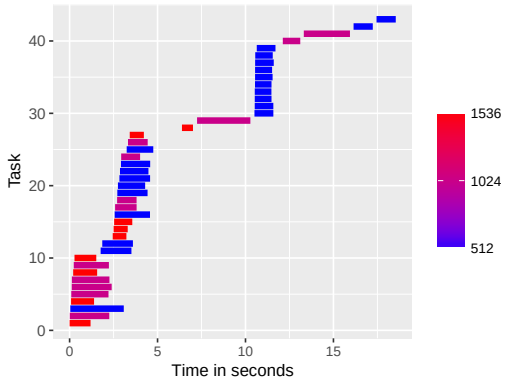
The resource types that have been assigned to each task are represented in Figures 3a, 3b and 3c. As we can see the workflow's execution accomplishes within the deadline constraint, and in some cases within the budget constraint, according to the assumptions of the algorithm.

Table 2 compares the execution scheduled by SDBWS with the homogeneous resource on which the execution took longer. Taking the experiment with $a_D = 0.1, a_B = 0.3$ parameters into consideration, we can see that the proposed heterogeneous schedule is 33.3% faster than the execution of resource with 512MB memory size. However it is only 7.9% more expensive. Using the scheduling algorithm we can fit into the deadline and budget constraints which would not be possible by using only homogeneous resources. We can also observe in Figure 2 that the differences in cost are smaller than the differences in execution time. This is explained by the fact that faster functions complete the tasks in a shorter time which cancels out the effect on their higher price per time unit.

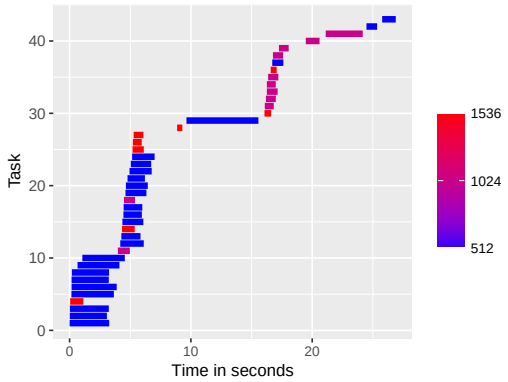
6. Conclusions and Outlook

In this paper, we discussed the challenges related to scheduling scientific workflows onto cloud functions. Although the serverless execution model automates most of the resource management tasks, the decisions regarding the selection of cloud function types (size) remain with the user and can be used by scheduling algorithms. We have shown this on the example of the SDBWS algorithm we developed, by adapting the existing DBWS heuristic from the IaaS to the FaaS model. We have also observed that taking into account the overheads is crucial for obtaining usable results.

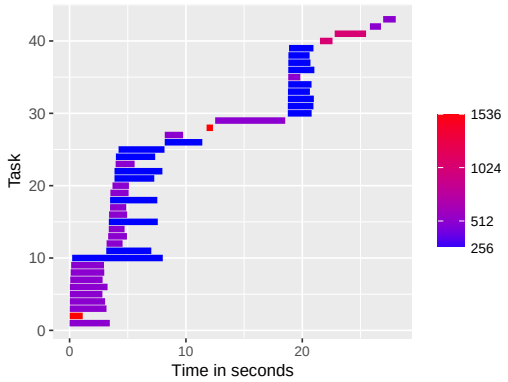
Our results of evaluation are preliminary, but we can draw some interesting conclusions from them. First, correct selection of resources for scheduling workflow tasks has significant impact on the performance and cost. Simply executing workflows on homogeneous resources may lead to violating the deadline or budget constraints. Choosing heterogeneous resources is a non trivial problem, and we can observe that while SDBWS does meet the deadline



(a) $a_D = 0.1$ and $a_B = 0.3$



(b) $a_D = 0.3$ and $a_B = 0.7$



(c) $a_D = 0.7$ and $a_B = 0.3$

Figure 3: Execution trace showing resources assigned to each task

constraint, it may fail to meet the budget constraint. This depends on the values of these constraints, since tightening the constraints makes the problem more difficult.

It is also important to take into account the overheads introduced by cloud providers. In our implementation of SDBWS we include the overheads measured during test runs to use the actual makespan for estimating the sub-deadlines. We observe that the real execution takes more time than the theoretical makespan calculated assuming that there are no

overheads.

There are several known limitations of our approach. SDBWS is a static scheduling algorithm, so it assumes the knowledge of task execution times on all resource types. This limits its applicability to workflows that are run repeatedly on some regular basis (e.g. daily). An alternative approach would be to use dynamic scheduling algorithms that can make decisions at runtime. We also assume that the data transfer time is included in the task runtime. Better approaches should model data transfers separately. Finally, SDBWS assumes that the entire workflow can run on the FaaS infrastructure, so there is no capability of off-loading larger tasks to IaaS.

As this is a report from an on-going work, we plan to run more experiments with other larger-scale workflows and with a wider range of deadline and budget parameters, to study the challenges we discussed in this paper. We plan also to extend our algorithm to handle hybrid FaaS-IaaS scenarios, model overheads and data access architecture. Future work includes also a comparison to other heuristics and more evaluation metrics to assess various scheduling approaches.

Acknowledgments

We would like to thank the anonymous reviewers from the Serverless Computing workshop for their suggestions on how to improve the paper.

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

References

[1] M. Malawski, A. Gajek, A. Zima, B. Balis, and K. Figiela, "Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions," vol. (In Print), 2017.

[2] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Serverless execution of scientific workflows," in *Service-Oriented Computing. ICSOC 2017*, M. Maximilien, A. Vallecillo, J. Wang, and M. Oriol, Eds., vol. 10601 LNCS. Springer, Cham, nov 2017, pp. 706–721. [Online]. Available: http://link.springer.com/10.1007/978-3-319-69035-3_51

[3] M. Malawski, K. Figiela, A. Gajek, and A. Zima, "Benchmarking heterogeneous cloud functions," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, D. B. Heras and L. Bougé, Eds., vol. 10659 LNCS. Springer, aug 2018, pp. 415–426. [Online]. Available: http://link.springer.com/10.1007/978-3-319-75178-8_34

[4] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "Serverless computing for container-based architectures," *Future Generation Computer Systems*, vol. 83, pp. 50–59, jun 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17316485

[5] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, fast and slow: Low-latency video processing using thousands of tiny threads," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 363–376. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi

[6] M. Ghasemzadeh, H. Arabnejad, and J. G. Barbosa, "Deadline-Budget constrained Scheduling Algorithm for Scientific Workflows in a Cloud Environment," in *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), P. Fatourou, E. Jiménez, and F. Pedone, Eds., vol. 70. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 19:1–19:16. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/7088

[7] A.-M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 351–359. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.32

[8] S. Nadgowda, S. Suneja, and A. Kanso, "Comparing scaling methods for linux containers," in *Proceedings - 2017 IEEE International Conference on Cloud Engineering, IC2E 2017*. IEEE, apr 2017, pp. 266–272. [Online]. Available: http://ieeexplore.ieee.org/document/7923811/

[9] B. Balis, "Hyperflow: A model of computation, programming approach and enactment engine for complex distributed workflows," *Future Generation Computer Systems*, vol. 55, pp. 147 – 162, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X15002770

[10] B. Balis, K. Figiela, M. Malawski, M. Pawlik, and M. Bubak, "A lightweight approach for deployment of scientific workflows in cloud infrastructures," in *Parallel Processing and Applied Mathematics, 11th International Conference, PPAM 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 9573. Springer, 2016, pp. 281–290.

[11] E. van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A SPEC RG Cloud Group's Vision on the Performance Challenges of FaaS Cloud Architectures," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering - ICPE '18*. New York, New York, USA: ACM Press, 2018, pp. 21–24. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3185768.3186308

[12] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, mar 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6471969/

[13] J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. Prince, and Others, "Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking," *International Journal of Computational Science and Engineering*, vol. 4, no. 2, pp. 73–87, 2009.