Execution of scientific workflows on hybrid infrastructure, a case study of AWS Elastic Container Service in combination with AWS Lambda

M. Pawlik, B. Baliś, M. Malawski, M. Orzechowski, K. Pawlik

Presentation plan

1. The idea
2. Introduction to scientific workflows
3. Workflow execution engine
4. Studied infrastructures
    a. containers
    b. cloud functions
5. Hybrid infrastructure support - implementation
6. Experiment
7. Conclusion
8. References

The idea

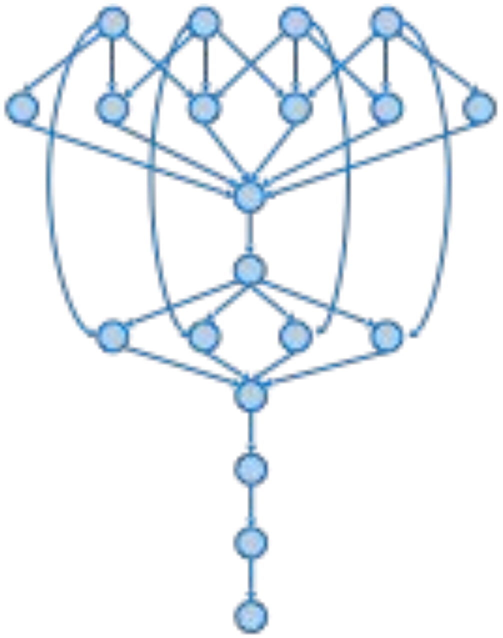
- Execution of scientific workflows on hybrid infrastructure...
- Some scientific workflows can benefit from being executed on *hybrid infrastructure*
  - In this case *Hybrid infrastructure* is achieved by combining two of *elastic* infrastructures containers with cloud functions

- The concept was implemented by extending an existing set of tools used to execute workflows [3] [4]
  - new Cloud Function component for Workflow Execution Engine
- Validate by experiment

Workflows

- high level description of the process
- graph representation
- dependency modeling

- decouple experiment from the infrastructure
- improved reusability/reproducibility
- supports parallelization



HyperFlow

- Simple, yet powerful, workflow execution engine [3]
    - aims to be an abstract workflow execution engine
- Written in JavaScript, uses Node.js runtime, some of the features:
    - lightweight
    - extensible
    - easy to understand and debug

- Doesn’t include support for execution on remote infrastructures
    - this can be achieved through extensions: *Functions* and *Executors*
        - function represents a given infrastructure on HyperFlow side
        - Executor is responsible for performing the operation

The elastic infrastructures

- Highly elastic cloud infrastructures:
    - on demand, almost instantaneous infrastructure provisioning
    - dynamic billing model
    - provide usable computing power
    - no/little system management
        - most of the management is automated or done through ‘infrastructure as a code’ concept

- Examples of such infrastructures: containerized environments, cloud functions

Containers

- Study of running workflows in containerized environment:
  M. Orzechowski, B. Baliś. "Container-Based Architecture for Resilient and Reproducible Scientific Workflows." CGW Workshop'17
  - targeted at container environments, namely: Kubernetes, AWS ECS etc.

- Promising approach, provides:
  - infrastructure as a code paradigm (by using Terraform)
  - easy infrastructure management
  - autoscaling
  - portability
- Some limitations:
  - Infrastructure indirectly operates on actual VMs
  - Adding new resources isn’t easy

Cloud functions

- Novel offering from cloud providers
- Developer prepares application or application components in form of source code for a given runtime
  - node.js, java, python etc.
  - implements a single *handler* function
- Cloud provider is responsible for infrastructure/resource provisioning
  - only one configurable parameter: memory size, also impacts available performance
- Function (application) instances are created on demand
  - so called *cold start* of a function is around 1 second

- callable through: REST API, messaging (AWS SQS), other events

Cloud functions cont.

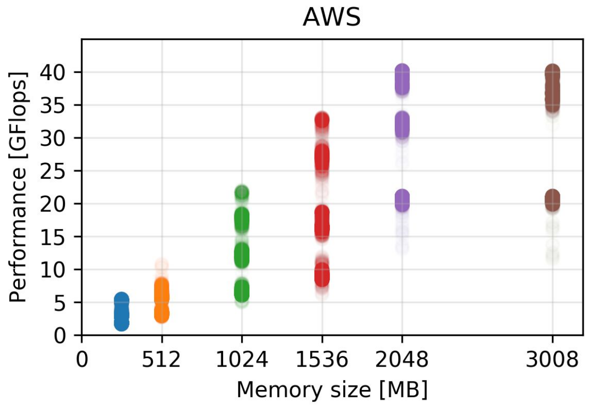
- a step further when it comes to elasticity, than containers or PaaS

- Functions provide:
    - Almost instant provisioning of new resources
    - 100ms billing granularity, function’s start overhead is not billed
    - massive parallelism, AWS allows up to 1000 simultaneous executions

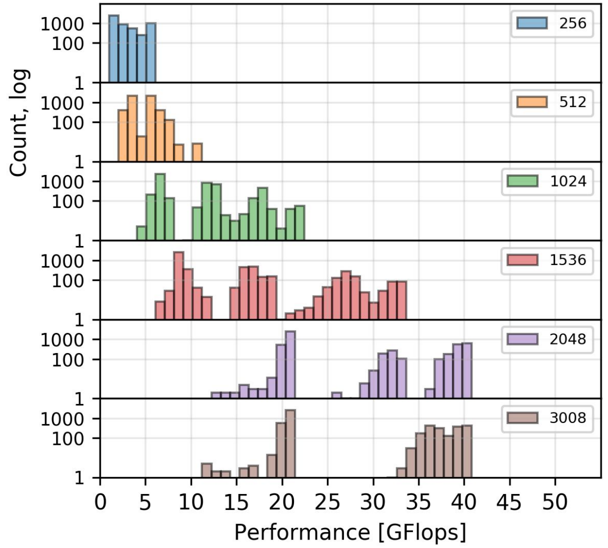
- Limitations (compared to containers):
    - limits on run time and memory (15 mins and 3GB at this point)
    - although miniscule, there is still an aspect of ‘cold start’
    - reliability
    - lack of environment reusability

Cloud function benchmarking

- Performance changes in relation to memory size
- In most cases there is one dominant value
- More information in [6] [7] [8]



- Ongoing work studying other aspects like:
  - provisioning speed
  - reliability
  - limits of parallelism

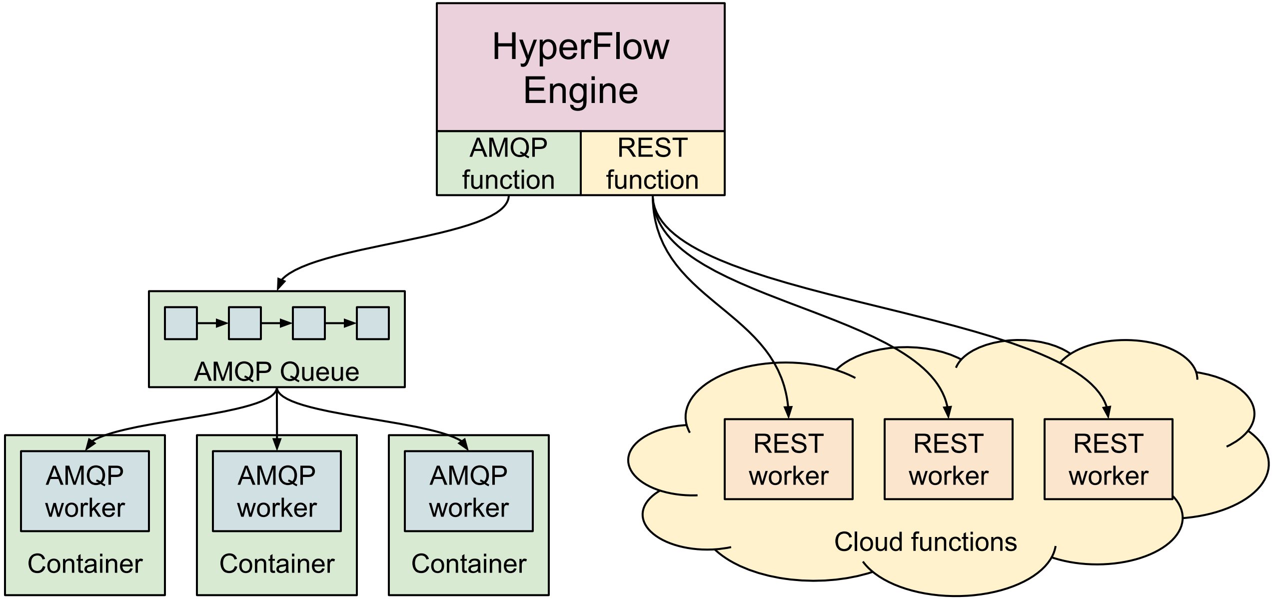


Implementation details

- Container execution uses existing “AMQP Command” Function
- Cloud functions use new “REST Service Command” function
  - HTTP used as a transport layer
  - compatible with multiple executors (exposing a proper API)
- Target infrastructure for each task is chosen before the start
  - Instrumentation is part of workflow graph

- Infrastructure as a code paradigm was implemented by using Terraform

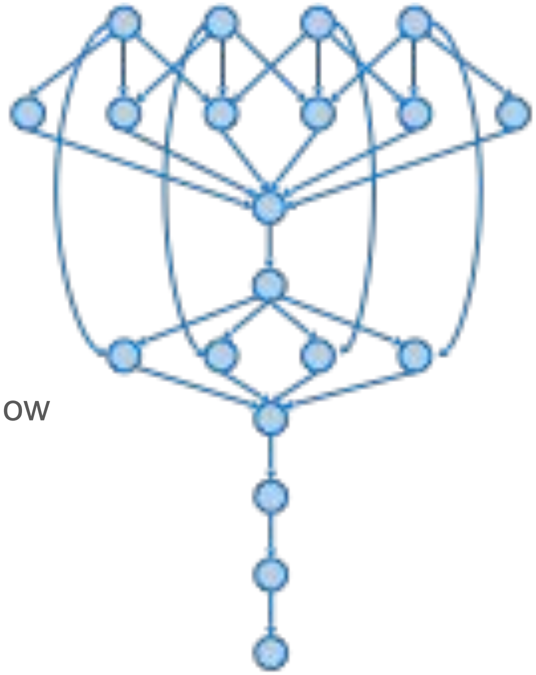
Architecture



Experiment setup

- Test two infrastructure models:
  - Container based
  - Hybrid (containers + cloud functions)

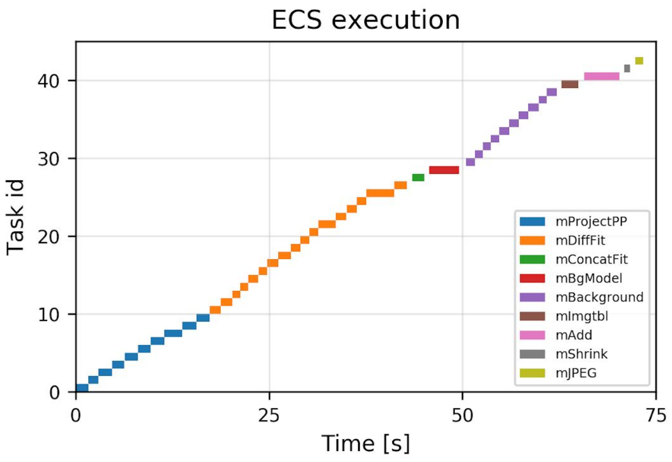
- Testing workload:
  - Montage (Image Mosaic Software), a popular example of workflow
  - 0.5 degree workflow
  - over 40 tasks
    - parallelizable

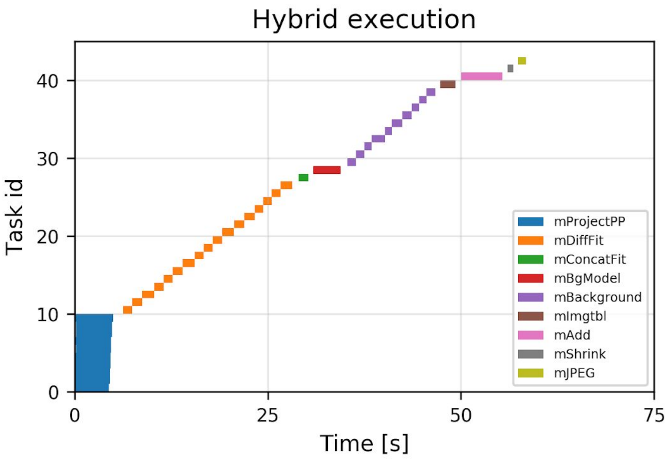


Experiment results (preliminary)

- Left chart: single Instance of container-worker hosted by t2.micro managed through ECS
- right chart: above infrastructure was extended with 256 MB Lambda

- Overall timespan was reduced, but at a slightly higher cost of execution





Hybrid Infrastructure scheduling

- Build on a basis of “Cloud function optimizer” [8]
  - workflow preprocessor
  - builds an execution plan, determines target infrastructure and its configuration based on certain criteria
- Prepare execution plan based with constraints like:
  - cost
  - timespan/deadline
  - data access overheads

Conclusions

- Using hybrid infrastructure can be beneficial for certain type of workflows
- Cloud functions can't be treated as universal replacement for containers, rather a supplement
- Overall hybrid infrastructure allows for covering a wider space of possible workflow execution constraints

References

1. Deelman, Ewa, et al. "Workflows and e-Science: An overview of workflow system features and capabilities." Future generation computer systems 25.5 (2009): 528-540.
2. Malawski, Maciej. "Towards Serverless Execution of Scientific Workflows-HyperFlow Case Study." Works@ Sc. 2016.
3. Balis, Bartosz. "HyperFlow: A model of computation, programming approach and enactment engine for complex distributed workflows." Future Generation Computer Systems 55 (2016): 147-162.
4. Orzechowski, Michał, and Bartosz Baliś. "Container-Based Architecture for Resilient and Reproducible Scientific Workflows." CGW Workshop'17 : proceedings
5. Berriman, G. Bruce, et al. "Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand." Optimizing Scientific Return for Astronomy through Information Technologies. Vol. 5493. International Society for Optics and Photonics, 2004.
6. Malawski, Maciej, et al. "Benchmarking Heterogeneous Cloud Functions." European Conference on Parallel Processing. Springer, Cham, 2017.
7. Pawlik, Maciej, Kamil Figiela, and Maciej Malawski. "Performance evaluation of parallel cloud functions." (2018).
8. Kijak, Joanna, et al. "Challenges for Scheduling Scientific Workflows on Cloud Functions." 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE, 2018.
9. **HyperFlow WMS github organization: https://github.com/hyperflow-wms/**