Challenges for Scheduling Scientific Workflows on Cloud Functions

Joanna Kijak, Piotr Martyna, Maciej Pawlik, Bartosz Balis and Maciej Malawski

Department of Computer Science,
AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Kraków, Poland

http://dice.cyfronet.pl
Outline

- Motivation: scientific workflows in clouds
- Experiments with HyperFlow
- Scheduling challenges
- Experiments with SDBWS algorithm
- Results on AWS Lambda
- Conclusions
DICE Team

- Investigation of methods for building complex scientific collaborative applications
- Elaboration of environments and tools for e-Science
- Integration of large-scale distributed computing infrastructures
- Knowledge-based approach to services, components, and their semantic composition

AGH University of Science and Technology (1919)
- 16 faculties, 36000 students; 4000 employees

Academic Computer Centre CYFRONET AGH (1973)
- 120 employees

Faculty of Computer Science, Electronics and Telecommunications (2012)
- 2000 students, 200 employees
- [http://www.iet.agh.edu.pl/](http://www.iet.agh.edu.pl/)

Department of Computer Science AGH (1980)
- 800 students, 70 employees
- [http://www.ki.agh.edu.pl/uk/index.htm](http://www.ki.agh.edu.pl/uk/index.htm)
Motivation: Scientific Workflows

- Astronomy, Geophysics, Genomics, Early Warning Systems...
- Workflow = graph of tasks and dependencies, usually directed acyclic graph (DAG)
- Granularity of tasks
  - Large tasks (hours, days)
  - Small tasks (seconds, minutes)
Infrastructure – from clusters to clouds

- Traditional HPC clusters in computing centers
  - Job scheduling systems
  - Local storage

- Grids to Clouds
  - Infrastructure as a service
  - Globally distributed
  - Virtual machines (VMs)
  - On-demand
  - Cost in $$ per time unit
Workflow execution model in (traditional) clouds

- Workflow engine manages the tasks and dependencies
- Queue is used to dispatch ready tasks to the workers
- Worker nodes are deployed in Virtual Machines in the cloud
- Cloud storage such as Amazon S3 is used for data exchange
- Examples
  - Pegasus, Kepler, Triana, Pgrade, Askalon, ...
  - HyperFlow (AGH Krakow)
HyperFlow
Lightweight workflow programming and execution environment developed at AGH

Programming language ecosystem

Text editor (favorite IDE, or generating script)

Executable workflow description (JSON)

hflowc run <workflow_dir>

Running a workflow – simple command line client

<workflow_dir> contains:

- File workflow.json (wf graph)
- File workflow.cfg (wf config)
- Optionally: file functions.js (advanced workflow activities)
- Input files

New challenges – serverless architectures

- **Serverless** – no traditional VMs (servers)
- **Composing of applications from existing cloud services**
  - Typical example: web browser or mobile device interacting directly with the cloud

**Examples of services:**
- Databases: Firebase, DynamoDB
- Messaging: Google Pub/Sub
- Notification: Amazon SNS

**Cloud Functions:**
- Run a custom code on the cloud infrastructure
Cloud Functions – good old RPC?

- **Examples:**
  - AWS Lambda
  - Google Cloud Functions (beta)
  - Azure Functions
  - IBM Bluemix OpenWhisk

- **Functional programming approach:**
  - Single function (operation)
  - **Not** a long-running service or process
  - Transient, stateless

- **Infrastructure (execution environment) responsible for:**
  - Startup
  - Parallel execution
  - Load balancing
  - Autoscaling

- **Triggered by**
  - Direct HTTP request
  - Change in cloud database
  - File upload
  - New item in the queue
  - Scheduled at specific time

- **Developed in specific framework**
  - Node.js, Java, Python
  - Custom code, libraries and binaries can be uploaded

- **Fine-grained pricing**
  - Per 100ms * GB (Lambda)
Earlier results and scheduling problem

- HyperFlow on Serverless:
  - AWS, Google, IBM

- Benchmarking of cloud functions:
  - AWS, Google, Azure, IBM

- M. Malawski, A. Gajek, A. Zima, and K. Figiela. Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions, FGCS 2018
- K. Figiela, A. Gajek, A. Zima, B. Obrok, M. Malawski: "Performance Evaluation of Heterogeneous Cloud Functions", Concurrency and Computation Practice Experience, 2018 (accepted)
- [http://cloud-functions.icsragh.edu.pl/](http://cloud-functions.icsragh.edu.pl/)
Scheduling challenges

- Resource selection: which task on which cloud function type?
- Hybrid execution: which task on FaaS and which on IaaS?
- How to deal with performance variability of infrastructure?
- What are the limits of concurrency that we can expect?
- How to transfer data between tasks?
Serverless Deadline-Budget Workflow Scheduling

- Adaptation of existing DBWS heuristic for serverless model
  - Low complexity heuristic based on PEFT
  - Uses VM model with hourly billing
- List scheduling heuristic algorithms, two phases:
  - Task ranking / prioritization (not used here)
  - Resource selection
- Assumes the knowledge of task runtime estimates on each resource type
- Finds mapping between tasks and resources (cloud functions) to meet the deadline constraint and tries to meet the budget constraint


Step 1: Levels

Divide DAG into levels
Step 2: Task runtime estimates

- Prerequisite to scheduling
- Run the workflow on all resource types
- Homogenous execution:
  - Function 1 is faster
  - Function 2 is slower
Levels and sub-deadlines

- For each workflow level compute the maximum execution time

$$Level_{execution}^j = \max_{l(t_i) = j} \{ ET_{\text{max}}(t_i) \}$$

- Divide the deadline into sub-deadlines proportionally for each level:

$$Level_{DL}^j = Level_{DL}^{j-1} + D_{user} \ast \frac{Level_{execution}^j}{\sum_{1 \leq j' \leq l(t_{exit})} Level_{execution}^{j'}}$$
Resource selection is based on the time and cost:

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

\[
Cost_Q(t_{cur}, r) = \frac{Cost_{max}(t_{cur}) - Cost(t_{cur}, r)}{Cost_{max}(t_{cur}) - Cost_{min}(t_{cur})} \cdot \xi
\]

- \(Time_Q\) – how far is task finish time on resource \(r\) from sub-deadline
- \(Cost_Q\) – how cheaper it is from the most expensive resource

\[
S_{DL}(t_{cur}) = \{Level_{DL}^j | l(t_i) == j\}
\]

\[
\xi = \begin{cases} 
1 & \text{if } FT(t_{cur}, r) < S_{DL}(t_{cur}) \\
0 & \text{otherwise}
\end{cases}
\]
We select the resource which maximizes the quantity:

\[ Q(t_{\text{cur}}, r) = T_{\text{ime}}(t_{\text{cur}}, r) \times (1 - C_F) + C_{\text{ost}}(t_{\text{cur}}, r) \times C_F \]

Where \( C_F \) is a trade-off factor:
- \( C_{\text{ost}_{\text{low}}} \) – cost on cheapest resource
- \( B_{\text{user}} \) – user’s budget

It represents user preferences:
- Lower value means we prefer to pay more for faster execution
- Higher value means we prefer cheaper and slower solutions

Idea:
- To finish as early as possible, and
- To find the cheapest resource
Sub-deadlines and resource allocation

- Result: heterogeneous execution
- Resource performance:
  - Function 1 is faster
  - Function 2 is slower
Schedule
Schedule
Schedule
Schedule
Schedule

The diagram represents a timeline with intervals and tasks labeled as t1, t2, t3, t4, t5, t6, and t7. The timeline is marked with time points at 0, 5, 8.5, 9.4, 12.24, 24.4, 30.4, 37.74, 42.84, 50.4, and 59.84.
Montage workflow, 43 tasks

Function size: 256, 512, 1024, 1536 MB

Execution times estimated based on pre-runs on homogeneous resources

Limits adjusted to fit between minimum and maximum measured values

Take into account the delays of task execution:
  – the makespan used to calculate the sub-deadlines includes all the overheads measured during pre-runs
Experiment 1

- Deadline: 18.6s (short)
- Budget: $0.00086 (small)

→ result: faster resources selected

*real* – AWS Lambda execution, *sdbws* – ideal case (no delays)
Experiment 2

• Deadline: 26.7s (medium)
• Budget: $0.00094 (large)

→ result: more slower resources selected
Experiment 3

- Deadline: 42.8s (large)
- Budget: $0.00086 (small)

→ result: slower resources selected
Conclusions

- Serverless and other highly-elastic infrastructures are interesting options for running high-throughput scientific workflows
- Serverless provisioning model are changing the game of resource management – but there still some decisions to make!
- Experiments with SDBWS show that heterogeneous execution may have advantages, but more tests are needed
- Cloud functions are heterogeneous
  - Technologies, APIs
  - Resource management policies (over/under provisioning)
  - Performance variations and guarantees
Future Work

- Evaluation of parallelism limits and influence of delays
- Combined FaaS-IaaS execution model
- Key parameter: elasticity
  - How quickly the infrastructure responds to the changes in workload demand
  - How fine-grained pricing can be?
  - Granularity of tasks vs. granularity of resources
- Example questions:
  - Which classes of tasks/workflows are suitable for such infrastructures?
  - How to dispatch tasks to various infrastructures?
  - How much costs can we actually save when using such resources (e.g. for tight deadlines/high levels of parallelism)?
Thank you!

- DICE Team at AGH & Cyfronet
  - Marian Bubak, Piotr Nowakowski, Bartosz Baliś, Tomasz Gubała, Maciej Pawlik, Marek Kasztelnik, Bartosz Wilk, Jan Meizner, Kamil Figiela

- Collaboration
  - USC/ISI:
    - Ewa Deelman & Pegasus Team
  - Notre Dame:
    - Jarek Nabrzyski

- Projects & Grants
  - National Science Center (PL)

- References:
  - HyperFlow: [https://github.com/dice-cyfronet/hyperflow/](https://github.com/dice-cyfronet/hyperflow/)
  - DICE Team: [http://dice.cyfronet.pl](http://dice.cyfronet.pl)
Backup slides
Detailed Google Cloud Functions Performance Results

- Functions often run much faster than expected.
- How often? About 5% times.
Cost analysis

- List price vs. price/performance
- Different models:
  - AWS – proportional
  - IBM – invariant
  - Google: mixed
- For Azure we assume 1024 MB
Cost analysis

**FIGURE 11** Price for cloud function per 100 millisecond depending on RAM. For Azure we assumed the cost of 1024MB.

**FIGURE 12** Costs for execution of single task in our integer performance benchmark, for all cloud function providers depending on RAM. For Azure we assumed the cost of 1024MB.

**FIGURE 13** Comparison of cost vs. execution time of single task in our integer performance benchmark, for all cloud function providers depending on RAM. For Azure we assumed the cost of 1024MB.

29% were handled with E5-2666 v3 (2.90GHz). Remaining requests were handled with E5-2676 v3 (2.40GHz) (5%) and E5-2670 v2 (2.50GHz) (1%).

Those CPUs support TurboBoost and some cores may be running at slightly higher frequency than base. We did not observe significant correlation between CPU model and function performance though.
References