COLONIES - COMPUTE CONTINUUMS ACROSS PLATFORMS

A PREPRINT

Johan Kristiansson
Department of Computer Science
RISE Research Institutes of Sweden
Luleå, Sweden
johan.kristiansson@ri.se

Thomas Ohlson Timoudas
Department of Computer Science
RISE Research Institutes of Sweden
Luleå, Sweden
thomas.ohlson.timoudas@ri.se

Henrik Forsgren
Department of Computer Science
RISE Research Institutes of Sweden
Luleå, Sweden
thomas.ohlson.timoudas@ri.se

March 28, 2023

ABSTRACT

Running AI/ML models in production is becoming widespread. At the same time, developing and maintaining AI workloads are becoming more difficult. In particular, most workloads are not portable and cannot easily be moved from one provider to another. Creating and operating fully automated end-to-end workflows across devices, edge, cloud adds even more complexity.

This paper presents a novel framework for running computational workload across heterogeneous platforms. Colonies is based on a loosely coupled microservice architecture where complex workflows are broken down in composable functions that are executed by independently deployable executors. Using a HTTP protocol, functions can be composed into declarative workflows in any computer language. The workflows are then executed across platforms by independently deployed executors running in the cloud, edge, devices, or even in web browser, creating compute continuums across platforms. Colonies support both real-time processing and batch jobs while at the same time offer full traceability and zero-trust security.

The paper also describes how Colonies can be used to build a scalable remote sensing platform on Kubernetes, how it can be used as a building block for edge computing, and how it can be integrated with HPC platforms. Finally, the paper presents a performance investigation as well as a scalability and robustness evaluation.

Keywords Serverless computing · Parallel computing · Workflow orchestration

1 Introduction

Building robust and scalable AI systems is challenging and requires deep understanding in various fields. First an AI model must be trained, requiring technical skills in advanced statistics or machine learning, but also access to training and validation data. Usually, the data need to be pre-processed in several steps before it can be used, or a simulator needs to be developed to either generate syntentic data or play back historical data. While it may be resonable for small scale projects to run a whole environment on a local development computer, training large AI models usually requires access to powerful compute clusters or even HPC systems. Manually utilizing such infrastructure is cumbersome and time consuming. Being able to automate the training processes makes it possible to more quickly iterate and find useful models.

Going forward and taking an AI model into production requires significant software engineering skills. In constract to traditional IT workloads both the data and the model itself need to be managed. As most models need to be re-trained or re-calibrated regularly, it muste be possible to seamlessly update deployed model and the software without loosing information or introducing delays. In many cases, there is a constant flow of data that is ingested into the system that needs to be managed while parts of the system is not working correctly. This becomes even more challenging when nodes parts of the underlying infrastructure crashes or becomes unavailable due to maintenance such as software updates or misconfiguration errors.

Sometimes there is also need to scale the system to increase capacity or scale down to save resources. This is particularly important when using expensive cloud resources. Scaling the system means that underlying infastructure may change anytime causing additional failures. It must therefor be possible to detected failed computations and re-process failed tasks part of a larger workflow. If failed computation cannot gracefully be recovered, there must be a away for engineers perform root cause analysis and manually recover failures.

Taking an AI model intro production,

Development - Data science - Production system - Integration, data ingestion, pre-processing Workflows, real-time processing, back processing. Operation - Computer faults - Updates, DevOps Scability HPC Requires a whole different set of toolchains.

https://modelserving.com/blog/why-do-people-say-its-so-hard-to-deploy-a-ml-model-to-production

2 The Colonies framework

2.1 Architecture

TODO

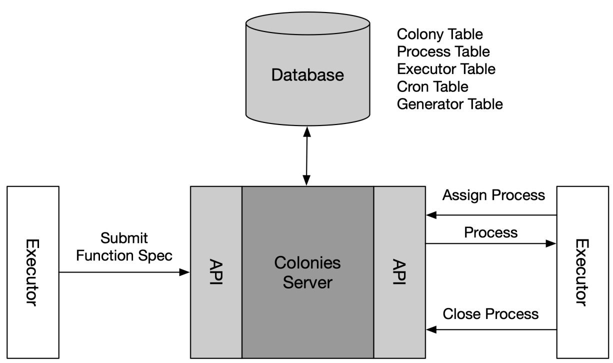

Figure 1: cron management

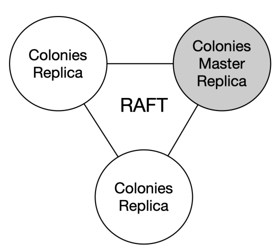

Figure 2: cron management

2.1.1 Workflows

TODO

dt = -1000000000 * 60 * 60 * 24 process.PriorityTime = int64(process.FunctionSpec.Priority)*dt + submissionTime.UnixNano()

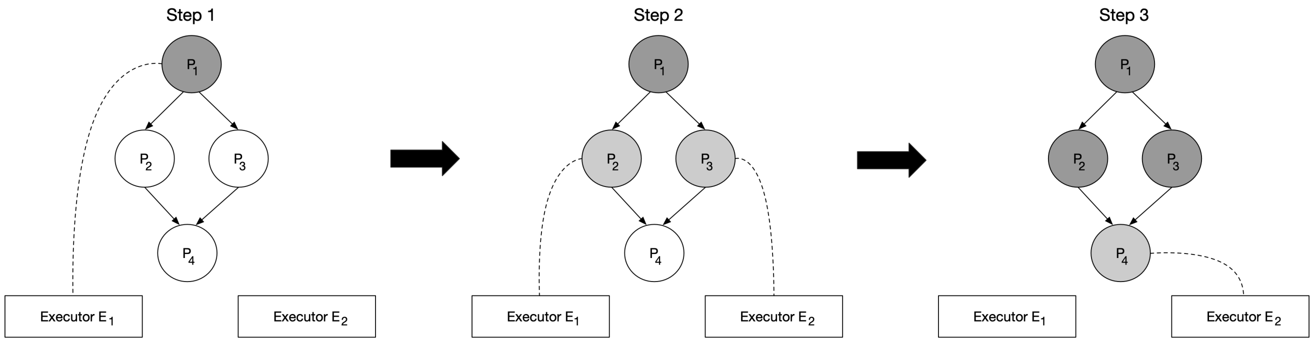

Figure 3: cron management

Table 1: Function Specifications

Function Spec	Function	Executor Type	Priority	Max Exec Time	Max Retries
F_1	gen_nums()	Edge	1	200 s	5
F_2	square()	Cloud	1	200 s	5
F_3	square()	Cloud	1	200 s	5
F_4	sum()	Browser	1	200 s	5

2.1.2 Cron

TODO

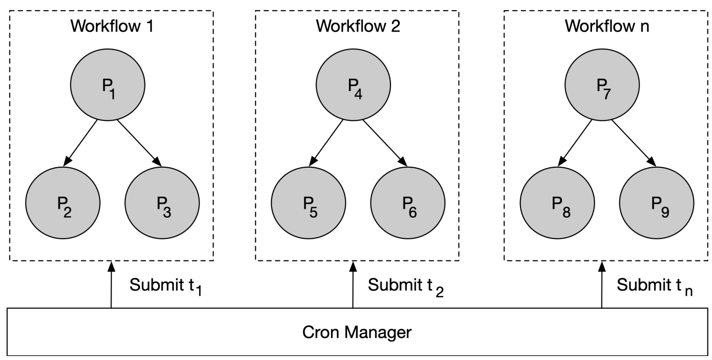

Figure 4: Sample figure caption.

2.1.3 Generators

TODO

2.1.4 Zero-trust security

TODO

3 Evaluation

3.1 Implementation

```
gen_nums = Function(gen_data, colonyid, executortype="edge")
square1 = Function(square, colonyid, executortype="cloud")
square2 = Function(square, colonyid, executortype="cloud")
sum = Function(square, colonyid, executortype="browser")
```

Table 2: Snapshot of Process Table as in Step 2

Process Id	Function Spec	Wait for Parents	Assigned Executor Id	State	Priority Time
P_1	F_1	$False$	E_1	Successful	1679906715352024000
P_2	F_2	$False$	E_1	Running	1679906715353453000
P_3	F_3	$False$	E_2	Running	1679906715354286000
P_4	F_4	$True$	-	Waiting	1679906715355188000

Table 3: Dependency Table

Process Id	Name	Dependencies
P_1	$Task_1$	-
P_2	$Task_2$	$Task_1$
P_3	$Task_3$	$Task_1$
P_4	$Task_4$	$Task_2, Task_3$

```
wf = ColoniesWorkflow("localhost", 50080, colonyid, executor_prvkey)
wf >> gennums
gennums >> square1
gennums >> square2
[square1, square2] >> sum
res = wf.execute()
```

3.2 References

TODO

Table 4: Input/Output Table

Process Id	Input	Output
P_1		[2,3]
P_2	2	4
P_3	3	9
P_4	[4,9]	13