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 29, 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.

In additional to a technical description, 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 a deep understanding of various fields. Firstly, an AI model must be trained, which requires technical skills in advanced statistics or machine learning, as well as access to training and validation data. Usually, the data needs to be pre-processed in several steps before it can be used, or a simulator needs to be developed to generate synthetic data or play back historical data. While it may be reasonable for small-scale projects to run the whole environment on local development computers, training larger AI models usually requires access to powerful compute clusters or even HPC systems. Manually utilizing such infrastructure is cumbersome and time-consuming. Automating the training processes makes it possible to iterate more quickly and find useful models.

Going forward and taking an AI model into production requires significant software engineering skills. In contrast 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 must be possible to seamlessly update the deployed model and the software without losing 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 are not working correctly. This becomes even more challenging when nodes or parts of the underlying infrastructure crash or become 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 instability problems for running applications. It must therefor be possible to detected failed computations and re-process failed tasks part of a larger workflow. Running workflows must therefor be designed to handle an ever changing infrastructure, and if a failed computation cannot gracefully be restored, there must be a away for engineers perform root cause analysis and manually recover failures.

In reality, several systems must be integrated to build fully working AI applications. For example, data might must be captured from an IoT system or pulled from third-party services running on different domains than the compute cluster. With the introduction of edge computing, parts of the data pipeline might also execute on edge servers, or even in on devices to bring computation closer where the data is produced. Configuring and setting up such pipelines adds further complexity.

In addition, many compute clusters run on-prem and somtimes there is a need to temporary increase the capacity by mixing on-prem and cloud resources, for example to handle peek load or re-process historical data. This also requires integration of various security protocols. Creating hybrid workflows where some jobs run on HPC systems requires even more software development efforts and is out of the scope for many users, preventing them from utilizing powerful hardware. Clearly, there is a need for a framework that can consolidate various platforms to simplify development and allowing computations to seamlessly run across platforms.

This paper aims at create a loosly coupled architecutre that seperates workflow definitions from implementation and deployment. The ultimate goal is to create an architecuture where monolitic workflows are broken down in independent compute units, which can be dynamically added to serve a workflow, and where the compute units could be implemented in any computer language and be deployed anywhere on the Internet. This is very similar to a distributed microservice cloud architecutre, but specifically targeting AI workloads. The next section describes a framework called Colonies

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

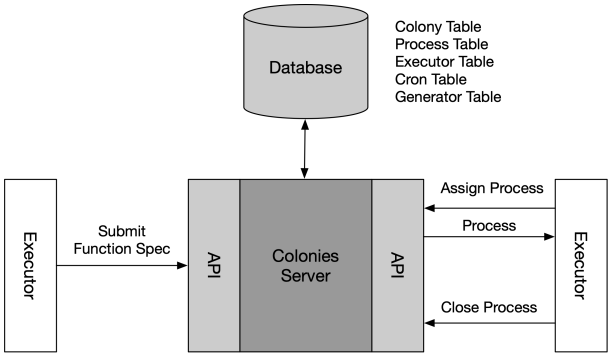

Figure 1: cron management

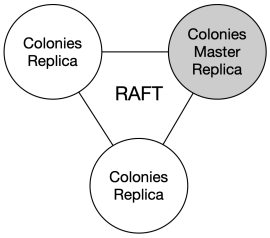

Figure 2: cron management

2.1.1 Workflows

TODO

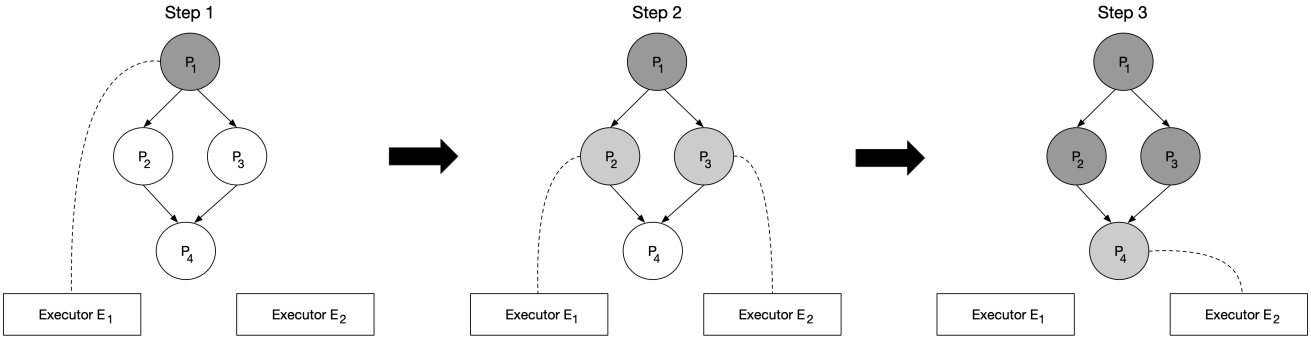

Figure 3: cron management

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

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

2.1.2 Cron

TODO

2.1.3 Generators

TODO

2.1.4 Zero-trust security

TODO

3 Evaluation

3.1 Implementation

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	$Task_4$	$Task_2, Task_3$

```
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")

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

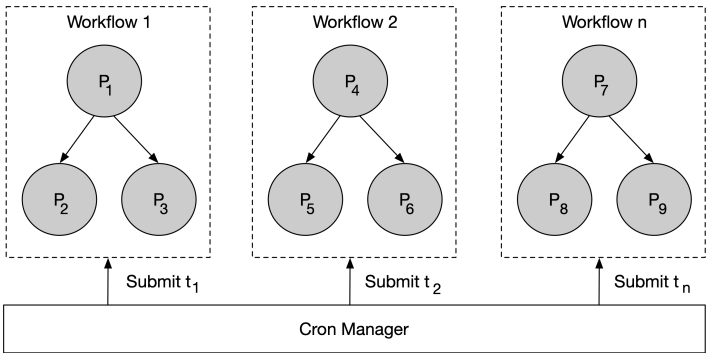

Figure 4: Sample figure caption.