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

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

March 29, 2023

ABSTRACT

Artificial intelligence and machine learning has gained significant traction in recent years. At the same time, development and operation of AI workloads has become increasingly challenging. One difficulty is the lack of portability, making it cumbersome to move from one platform or provider to another. Creating and operating fully automated end-to-end workflows across devices, edge, and cloud platforms is even more challenging.

To address these issues, the paper presents a novel framework called Colonies for running computational workloads across heterogeneous platforms. Colonies is built upon a loosely coupled microservice architecture that breaks down complex workflows into composable functions, which can be executed by independently deployed executors. With the use of an HTTP protocol, the functions can be composed into declarative workflows in any computer language, and be executed across platforms by executors running in the cloud, edge, devices, or even in web browsers, creating seamless compute continuums across platforms.

In addition to a technical description of the Colonies framework, the paper also describes some potential use cases. The paper describe how Colonies can be leveraged to build a scalable remote sensing platform on Kubernetes, serve 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.

In summary, Colonies is a highly versatile and scalable framework that can streamline the development and deployment of computational workloads across heterogeneous platforms while also ensuring full traceability and zero-trust security.

Keywords Serverless computing · Parallel computing · Workflow orchestration

1 Introduction

Developing robust and scalable AI systems is a challenging task that requires deep understanding in several fields. To begin with, an AI model must be trained which requires knowledge in advanced statistics or machine learning, as well as access to training and validation data. Typically, this data must be pre-processed through various stages before it can be utilized. Although it may be practical for small-scale projects to run the entire training processes on local development computers, larger AI models typically require access to powerful compute clusters or even HPC systems.

Manual use of such infrastructure can be laborious and time-consuming. Automating the training process enables faster iterations and quicker discovery of useful models.

Taking an AI model into production requires substantial software engineering expertise. In contrast to traditional IT workloads, both the data and the model must be managed in addition to the software itself. As most models require regular re-training or re-calibration, it must be possible to update deployed models and software seamlessly without losing information or breaking the system. In many cases, there is a constant flow of data ingested into the system which must be managed even if components are malfunctioning. This becomes even more challenging when nodes or parts of the underlying infrastructure become unavailable due to maintenance or failaure such as software updates, hardware replacements or simply missconfiguration.

In some cases, it may be necessary to scale the system to increase or reduce the capacity. This is especially critical when using expensive cloud resources. Scaling the system means that the underlying infrastructure may change at any time, causing instability issues for running services or workflows. Therefore, it must be possible to detect failed computations and reprocess failed tasks part of a larger workflow. Workflows must hence be designed to handle an ever-changing infrastructure, and if a failed computation cannot be restored gracefully, engineers must be able to quickly perform root cause analysis to manually recover the system.

In reality, AI system requires integration of multiple systems. For instance, data may need to be captured from an IoT system or pulled from third-party database running on different domains than the compute cluster. With the emergence of edge computing, parts of a data pipeline may also run on edge servers to bring computations closer to data sources. Configuring and setting up such pipelines add even more complexity.

Additionally, many compute clusters operate on-premises installations. Sometimes it is necessary to temporarily increase the capacity of on-prem clusters by combining resources from multiple providers, for example, adding cloud compute resources to handle peak loads or utilize HPC resources to quickly reprocess historical data. Developing hybrid workflows where some jobs run in the cloud and others run on HPC systems requires even more software development efforts and is beyond the scope of 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 enable seamless execution across platforms.

This paper presents a framework called Colonies, which is built around a loosely coupled microservices architecture that separates workflow definitions from implementation and deployment. The main objective is to create a tool where monolithic workflows can be broken down into independent and separated compute units that can be dynamically added or removed while executing workflows. These compute units can be implemented in any computer language and be deployed anywhere on the Internet. Ultimately, the Colonies framework facilitates coordination among diverse software systems operating on different platforms in executing unified workflows.

The next section presents related work. Section ....

2 Related work

A workflow is a series of tasks that need to be completed in a specific order. Workflows are often represented as directed acyclic graphs (DAGs), where nodes represent tasks and edges represent dependencies or data flow between tasks.

TODO:

3 The Colonies framework

Microservices is an architectural design pattern in which an application is structured as a collection of small, independently deployable, and loosely coupled services that communicate with other microservices through well-defined APIs. By dividing the application into smaller, focused microservices, applications become easier to understand, maintain, and develop. Each microservice can be scaled independently, making it easier to handle increased demand for a certain service. Additionally, different microservices can be developed by using diverse technologies, frameworks, or programming languages, enabling developers to select the most suitable tools for each specific problem. By assigning ownership of specific microservices to individual teams, it also becomes easier to coordinate work and maintain a consistent development process.

Currently, microservices are primarily used to implement large-scale web applications or Internet applications requiring high-availability. It has not yet become a prevalent design principle for workload management or implementation of HPC applications. Instead, simple job scripts are commonly used. In some cases, message brokers (e.g., RabbitMQ) are used to build worker queues to distribute tasks among multiple workers. Although this approach may be effective

for simpler applications, creating dependencies between tasks, such as controlling the order of execution or passing information between tasks is not straightforward.

Colonies is based on a microservice architecture where workflows are decomposed into a set of functions. An example of a function could for example be to train a neural network, prepare a batch of data, or upload inferred results to a third-party database. An executor is responsible for executing one or several functions, making it very simular to a microservice. All coordination is managed by a cluster of Colonies servers, allowing complex workflows to be broken down into independent functions that can be developed and tested separately. Analogous to traditional microservices, scalability can be achieved simply by adding more executors. If an executor crashes during task execution, the task is automatically reassigned to another executor.

3.1 The role of queues for seperation of concerns

Separation of concerns (SoC) is a fundamental design principle in computer science that aims to break down a complex software system into smaller, more manageable parts. For example, HTTP APIs can be used to abstract away implementation detail and provide a clear and simple interface for interacting with a particular service. However, HTTP protocols alone are insufficient for handling dynamic environments where components frequently fails or the underlying infrastructures is constantly changing.

To address such environments, an alternative mechanism is necessary. Queues enable different executors to communicate indirectly by acting as a buffer between them. Queues makes it possible to decouple each executor to operate independently, e.g. a service can be updated or replaced without affecting other executors. Queues also allow for asynchronous communication between executors, enabling them to process tasks at their own pace. This ensures that slower executors do not bottleneck faster ones, leading to a more efficient and scalable system. Most importantely, queues enable load balancing by distributing tasks among multiple executors, thus making it possible to parallelize workflow execution.

Queues can be implemented in various ways. As previously mentioned, a common solution involves using a message broker. However, Colonies adopts an alternative approach by utilizing a standard database and querying it to assign tasks to different executors. The advantage of employing a database lies in the fine-grained control it offers for assigning task to executors. For instance, it allows a specific tasks be assigned to particular executor type, thus enabling a workflow to be executed across diverse platforms.

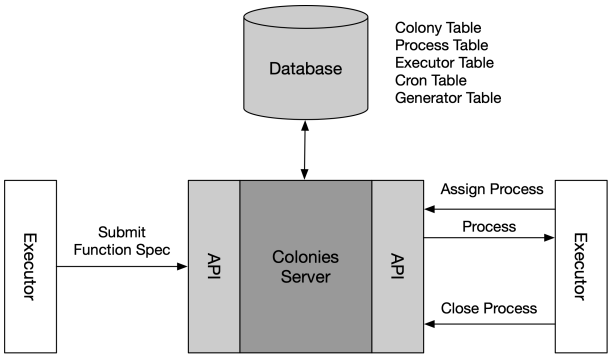

Figure 1: cron management

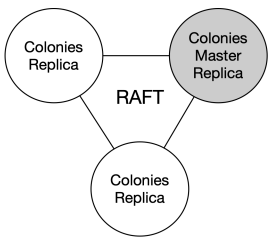

Figure 2: cron management

3.1.1 Workflows

TODO

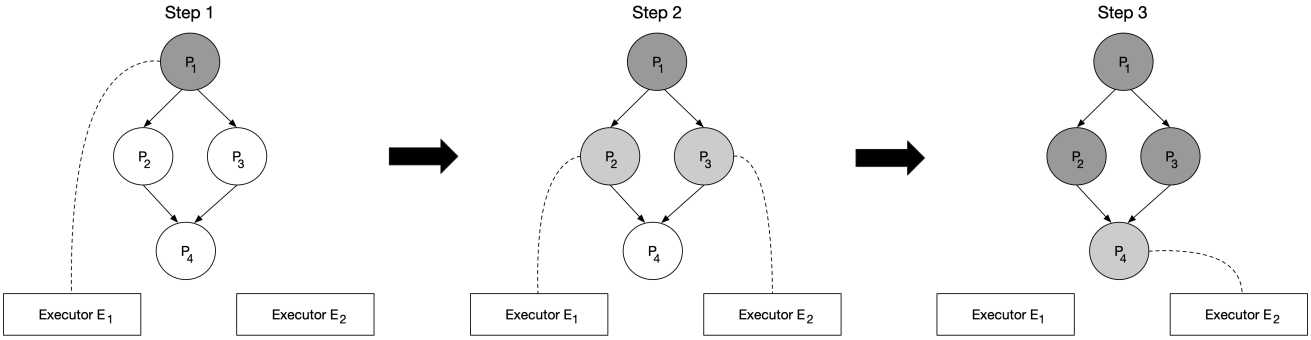

Figure 3: cron management

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

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

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

3.1.2 Cron

TODO

3.1.3 Generators

TODO

3.1.4 Zero-trust security

TODO

4 Evaluation

4.1 Implementation

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

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	$Task_4$	$Task_2, Task_3$

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

```
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()
```

4.2 References

TODO

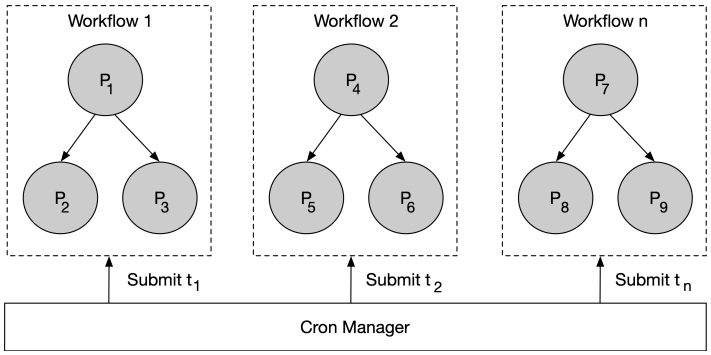

Figure 4: Sample figure caption.