Mixed Modality Workflows in TaskVine

David Simonetti
University of Notre Dame
South Bend, IN, USA
dsimone2@nd.edu

Ben Tovar
University of Notre Dame
South Bend, IN, USA
btovar@nd.edu

Douglas Thain
University of Notre Dame
South Bend, IN, USA
dthain@nd.edu

ABSTRACT

Modern scientific workflows desire to mix several different computing modalities: self-contained computational tasks, data-intensive transformations, and serverless function calls. To date, these modalities have required distinct system architectures with different scheduling objectives and constraints. In this paper, we describe how TaskVine, a new workflow execution platform, combines these modalities into an execution platform with shared abstractions. We demonstrate results of the system executing a machine learning workflow with combined standalone tasks and serverless functions.

ACM Reference Format:
David Simonetti, Ben Tovar, and Douglas Thain. 2023. Mixed Modality Workflows in TaskVine. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '23), June 16–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3588195.3595953

1 INTRODUCTION

Modern scientific workflows desire to mix several different computing modalities, which may include compute intensive applications on multicore CPUs, data-intensive applications which require substantial memory and disk, and latency-sensitive applications which require lightweight function calls. These application modes have typically separate distinct frameworks and resource management systems to serve their scheduling and management needs. This makes it challenge to construct multi-modal workflows, which require deploying multiple systems coordinated from one (or more) workflow management systems.

In this short paper, we demonstrate how TaskVine, a new workflow execution platform, enables the combined management of three different computing modalities. Traditional high throughput compute-intensive applications are expressed as **Tasks** that are scheduled primarily to allocate compute resources and rely on staged data. Data-intensive analysis tasks are expressed in the same way but scheduled primarily to take advantage of data locality and capacity within the cluster. Latency-sensitive tasks are expressed in a serverless style by first deploying a **LibraryTask** that is then invoked via later **FunctionCall**s that are scheduled around availablity of services. All three share a common underlying resource management framework that permits a workflow to use all three modalities and exchange data between them.

HPDC '23, June 16–23, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0155-9/23/06.
https://doi.org/10.1145/3588195.3595953

2 TASK AND FUNCTION ABSTRACTIONS

TaskVine is a workflow execution platform consisting of a manager process and a large number of worker processes that are deployed to the nodes of a compute cluster. The user writes a dynamic workflow through a Python API that defines data assets, standalone tasks, function libraries, and function invocations, and connects them together.

Standalone Tasks. A normal **Task** indicates a Unix program to be executed on a remote worker. As shown in Figure 1, a Task is executed in a private sandbox directory. Each Task is allocated a fixed quantity of resources and is monitored to ensure that it does not exceed the allocated amount, otherwise it is killed. The manager then schedules tasks to fill the available worker resources.

The standard Task modality is appropriate for compute intensive tasks that are relatively long-running (30 seconds or more). However, in many scientific workflows, similar tasks are executed many times with slight variations in input parameters. This can result in an duplication of the same initialization work, such as starting a container, loading common libraries, or reading a dataset from storage. In extreme cases, the initialization overhead can become a significant portion of the overall workflow if each task only runs for a short time. For example, we have observed cases where this overhead can grow to consume half of the total workflow execution time. Additionally, this problem is amplified on a shared file system where the same files get accessed repeatedly and can lead to overloading the file server.

Serverless Tasks. TaskVine extends the task abstraction into two specialized task types: LibraryTasks and FunctionCall tasks. A **LibraryTask** is a task which runs a **Library**, which is an arbitrary program containing a collection of functions which can be invoked by the worker. The manager installs a Library onto a worker by sending out a LibraryTask, which may be accompanied by dependent files and resources required to startup the Library. After receiving the LibraryTask, the worker forks a child process to run the Library. Once this is complete, the running Library process is called a **Library Instance** and it passively waits to receive messages from the worker. In order to invoke one of the functions on the Library, the worker uses the Library protocol, which involves sending a JSON invocation message describing the name of the function to execute and the arguments for that invocation.

FunctionCall tasks are used to perform these invocations. FunctionCall tasks specify the name of a function to run with serialized arguments. FunctionCall tasks are sent to a worker like normal tasks, but after the worker receives a FunctionCall, instead of invoking a Unix program, it communicates with the LibraryInstance using the Library protocol to invoke the specified function. Once the Library Instance receives the invocation, the Library Instance runs that function with the given arguments and returns the result back to the worker, and this is used as the output for the FunctionCall

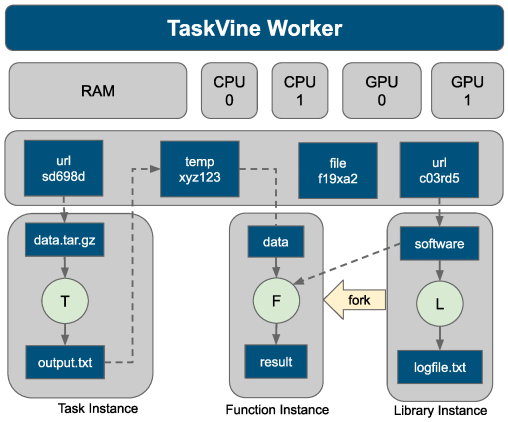

Figure 1: Combining Tasks and Functions at the Worker
TaskVine intermixes regular and serverless tasks in same resource management framework. A normal task T executes alongside a Library Instance L and FunctionCall F. All three types allocate shared resources, and consume and produce shared input and output objects.

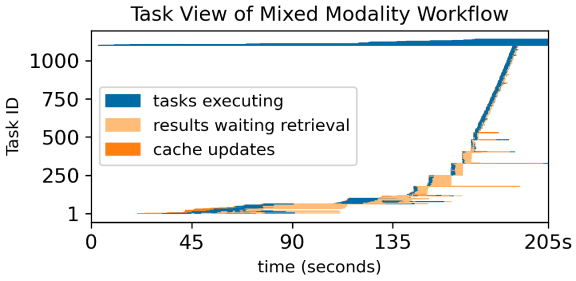

Figure 2: Combined Workflow Execution
Plot of mixed modality machine learning workflow. The tasks run in the first two minutes of the workflow consist of the longer-running standard Tasks training models. While this occurs, LibraryTasks are being deployed as can be seen by the large blue bar at the top of the graph. As Libraries are being deployed, FunctionCall tasks can begin to run; these are the smaller bars in the last minute of the workflow, and 1000 of these tasks can complete with minimal overhead.

Task. After completing the FunctionCall Task, the Library Instance returns to passively waiting for the next invocation request.

Mixed modality workflows are possible because all of the task types share the same underlying resource management methods, allowing them to co-exist. Like standard tasks, each Library instance consumes a static resource allocation on each worker defined by the LibraryTask. Each FunctionCall task also consumes a set of resources in addition to the LibraryTask. Tasks of all types can then be packed into a worker until all resources are consumed.

As a simple benchmark, we consider the cost of executing tasks that depend upon the Tensorflow library. A challenge of using Tensorflow workflow is its size and number of dependencies. A freshly created Conda environment with only Tensorflow installed measures 1.2GB in size and can take 5-10s simply to `import`. We compare a workflow of 500 null tasks consisting of nothing but `import tensorflow` executed on 50 2-core workers: the entire workflow completes in **61.86s** when 500 standard Tasks are used, but **11.69s** when defined as a single LibraryTask and 500 FunctionCalls. This experiment illustrates the high-level benefit that serverless tasks provide towards short-lived tasks with a large initialization cost. Furthermore, in the case where a workflow consists of both long-running and short-lived tasks, a combination of tasks types can be beneficial.

Figure 2 shows the timeline of a mixed modality workflow using TaskVine on 50 workers. 100 standard Tasks taking about 30s each are dispatched, each one requiring a 1.2GB Conda environment. The tasks utilize Tensorflow to train a 100,000 weight model on the MNIST handwritten digit dataset. LibraryTasks are deployed with a function that evaluates the model on a portion of the test data, and FunctionCall tasks are used to invoke these model assessments. These FunctionCall tasks occurs a total of 1000 times (10 per model), each one taking only a few seconds. As can be seen the system starts slowly, with the long-running training tasks that only pay a proportionally small initialization cost dominating the first two minutes of the workflow. Once the training tasks are completed, the serverless evaluation tasks begin to be deployed and are able to exploit the low initialization cost of individual serverless function calls. This allows these short-lived tasks to avoid continually rerunning expensive initialization tasks, resulting in an overall faster workflow and lower total resource usage. Overall, creating a mixed modality enables greater flexibility and resource savings by utilizing the upsides of both task types.

In the future, we intend to further explore the benefits that workflows combining serverless and regular tasks can provide. We intend to take preexisting scientific analysis applications and convert them to using a mixture of serverless and regular tasks and see if significant resource savings or runtime reductions follow.

3 RELATED WORK

OpenWhisk [3] is a serverless workflow system that focuses on reacting to external events with serverless invocations. OpenLambda [4] is a platform directed towards providing performance optimizations for severless workloads. Parsl [1] is Python library that allows users to easily create parallel scientific computing workloads. funcX [2] is a workflow execution platform which exclusively executes serverless workloads. Ray [5] is an AI focused workflow execution platform including a distributed scheduler.

Acknowledgement:
This work was supported in part by NSF grant OCI-1931348.

REFERENCES

[1] Y. Babuji. Parsl: Pervasive parallel programming in python. HPDC '19, New York, NY, USA, 2019. Association for Computing Machinery.
[2] R. Chard. Funcx: A federated function serving fabric for science. HPDC '20, New York, NY, USA, 2020. Association for Computing Machinery.
[3] T. A. S. Foundation. Apache openwhisk.
[4] S. Hendrickson. Serverless computation with openlambda. In *8th USENIX workshop on hot topics in cloud computing (HotCloud 16)*, 2016.
[5] P. Moritz. Ray: A distributed framework for emerging AI applications. In *OSDI*, 2018.