UNIVERSITY OF NOTRE DAME

Building High Throughput Function-Oriented Workflows with TaskVine

Douglas Thain and the CCL Team
University of Notre Dame
Throughput Computing 2024
Madison, WI July 2024

CCTools

Throughput Computing 2024

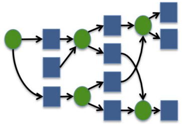
CHTC

How do I organize my work to use HTCondor?

https://condor.cse.nd.edu

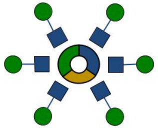
Makeflow

Unix-Oriented DAGs



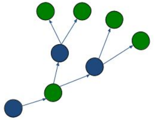
Work Queue

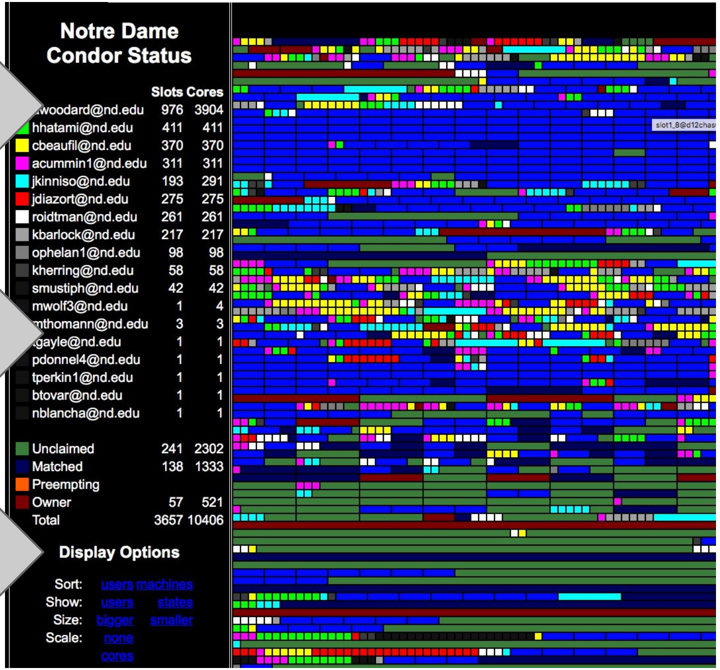
Dynamic Task Creation



TaskVine

Dynamic Data Sharing



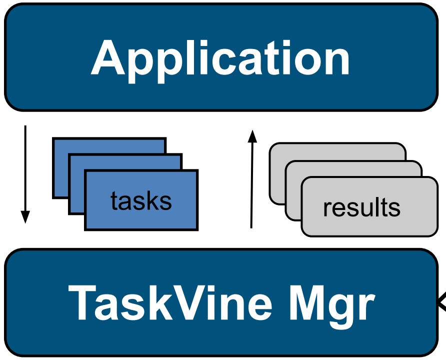


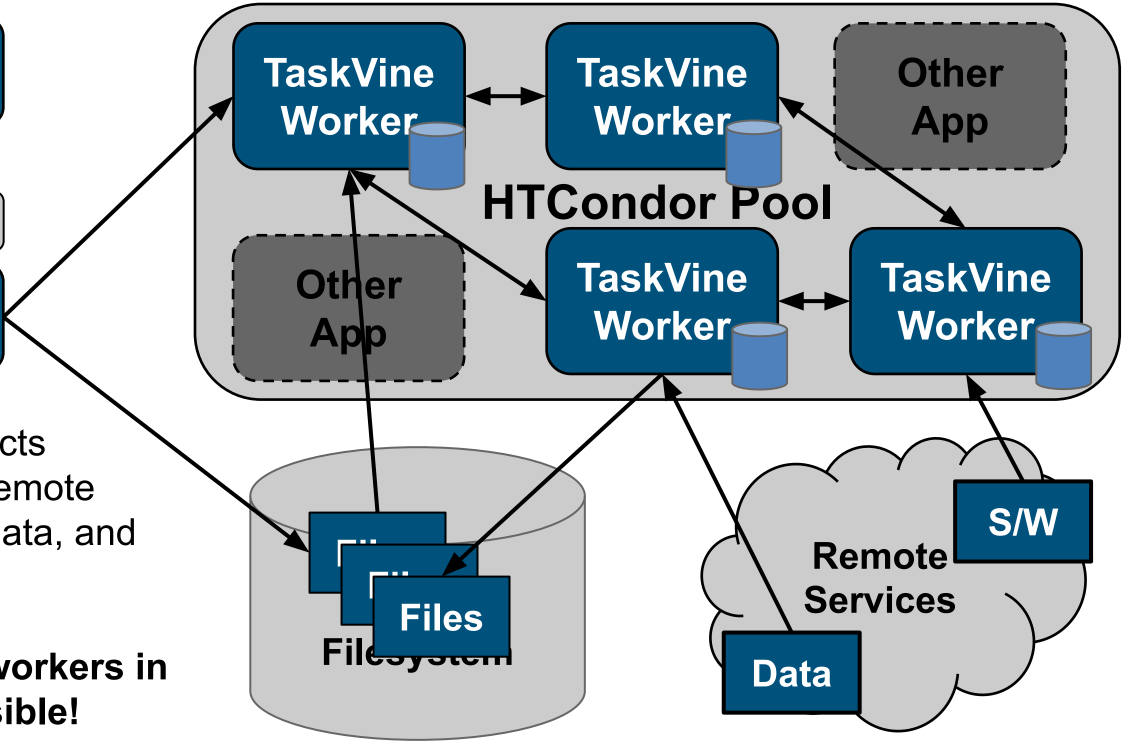
TaskVine

TaskVine is a system for executing **data intensive** scientific workflows on clusters, clouds, and grids from very small to massive scale.

TaskVine controls the computation **and storage** capability of a large number of workers, striving to carefully manage, transfer, and re-use data and software wherever possible.

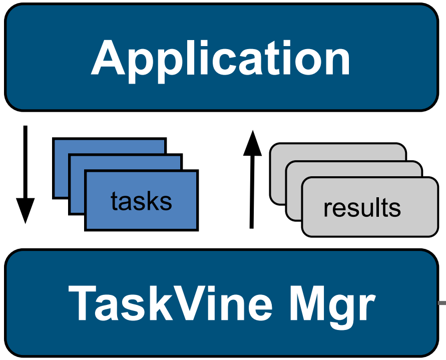
TaskVine Architecture Overview





The TaskVine manager directs workers to read data from remote sources, run tasks on that data, and share data with each other.

TaskVine leaves data on workers in the cluster wherever possible!

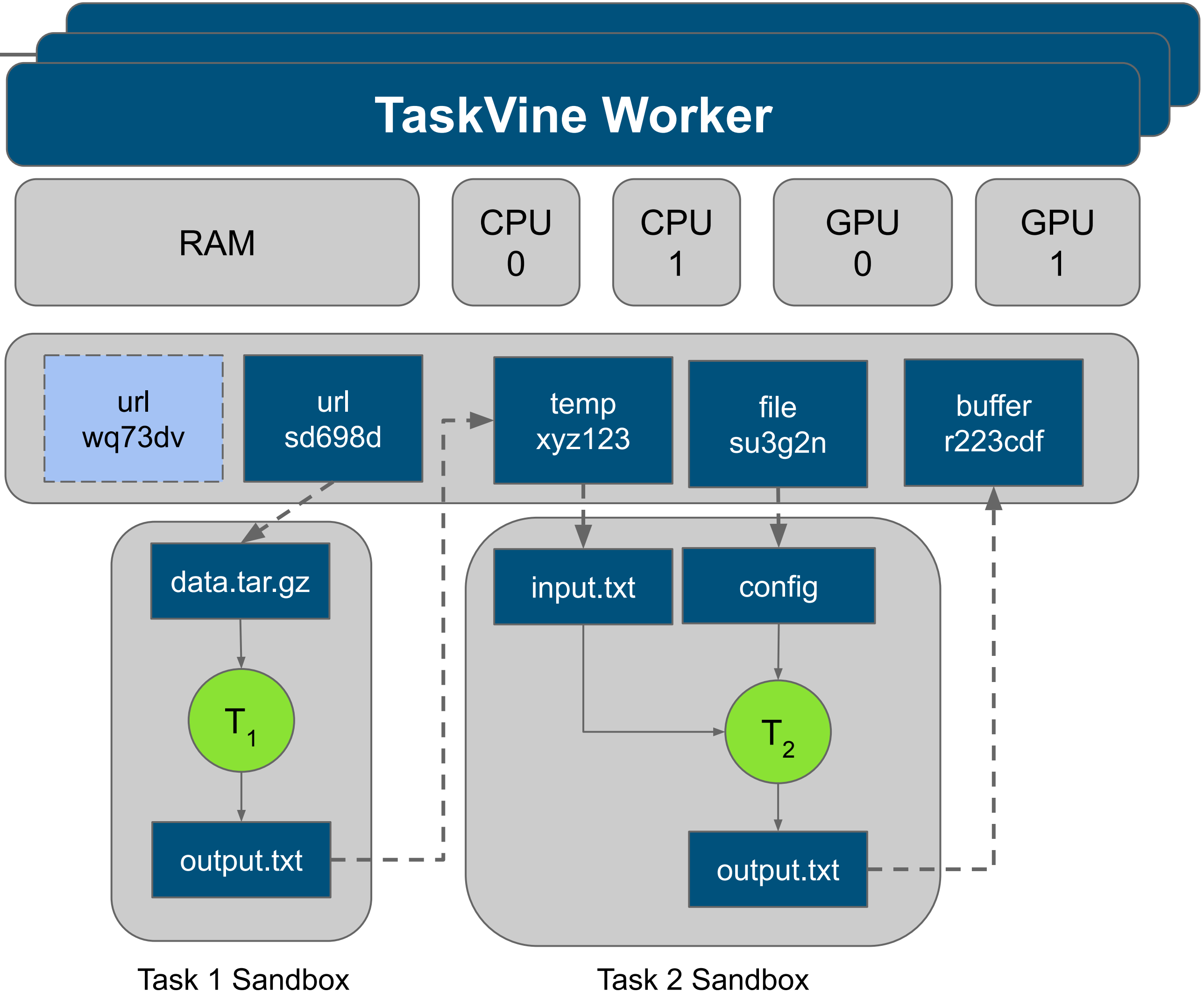


File = Single file or complex dir.

Manager directs all file movements and accesses.

Files are immutable and given a **unique cache name**.

Each task runs in a sandbox with a private namespace and an allocation of cores, memory, disk, and gpus.



API: Declare Files Explicitly

```
import ndcctools.taskvine as vine

m = vine.Manager(9123)

file   = m.declareFile("mydata.txt")
buffer = m.declareBuffer("Some literal data")
url    = m.declareURL("https://somewhere.edu/data.tar.gz")
temp   = m.declareTemp();

data     = m.declareUntar( url )
package  = m.declareStarch( executable )
```

API: Connect Tasks to Files

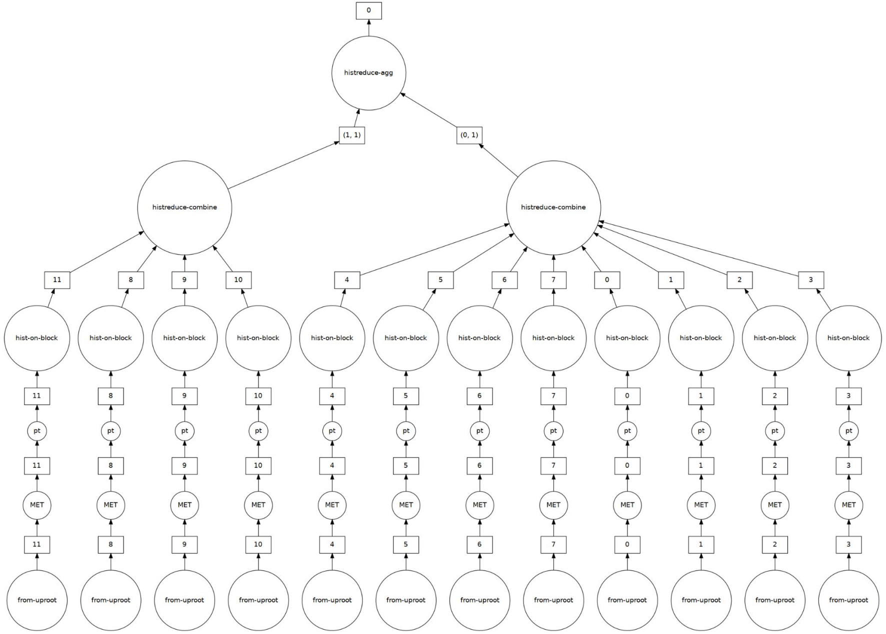
```
task = vine.Task("mysim.exe -p 50 input.data -o output.data")

t.add_input(url,"input.data")
t.add_output(temp,"output.data")

t.set_cores(4)
t.set_memory(2048)
t.set_disk(100)
t.set_tag("simulator")

taskid = m.submit(t)
```

API: Execute Python Function

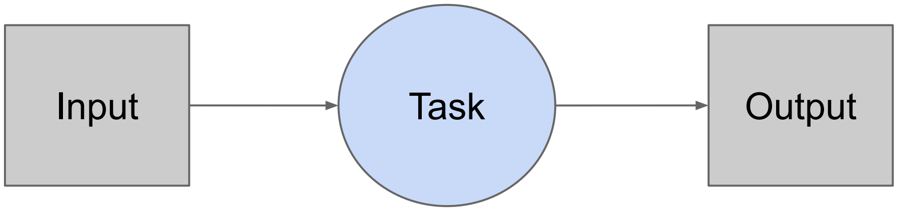
```
task = vine.PythonTask(simulate_func,molecule,parameters)

t.set_cores(4)
t.set_memory(2048)
t.set_disk(100)
t.set_tag("simulator")

taskid = m.submit(t)

. . .

print(t.result)
```

Building Up a Large DAG Manually

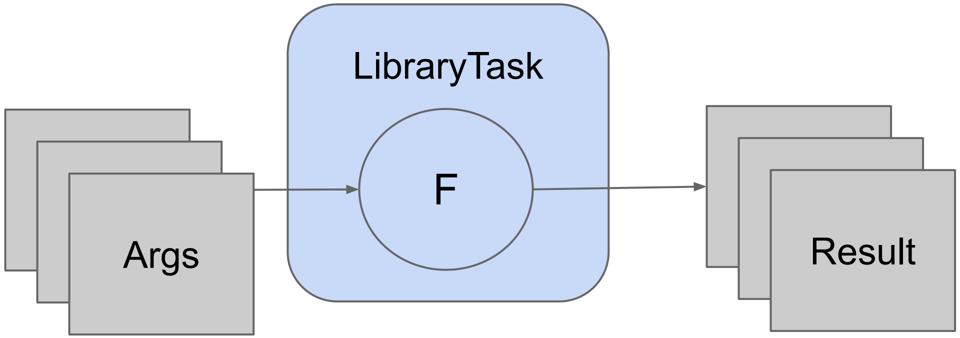
```
x = m.define_file()
y = m.define_file()
z = m.define_file()
. . .

a = Task()
b = Task()
c = Task()
. . .

a.add_input(x,"data")
a.add_output(y,"temp")
. . .
```



From Tasks to Libraries and Functions



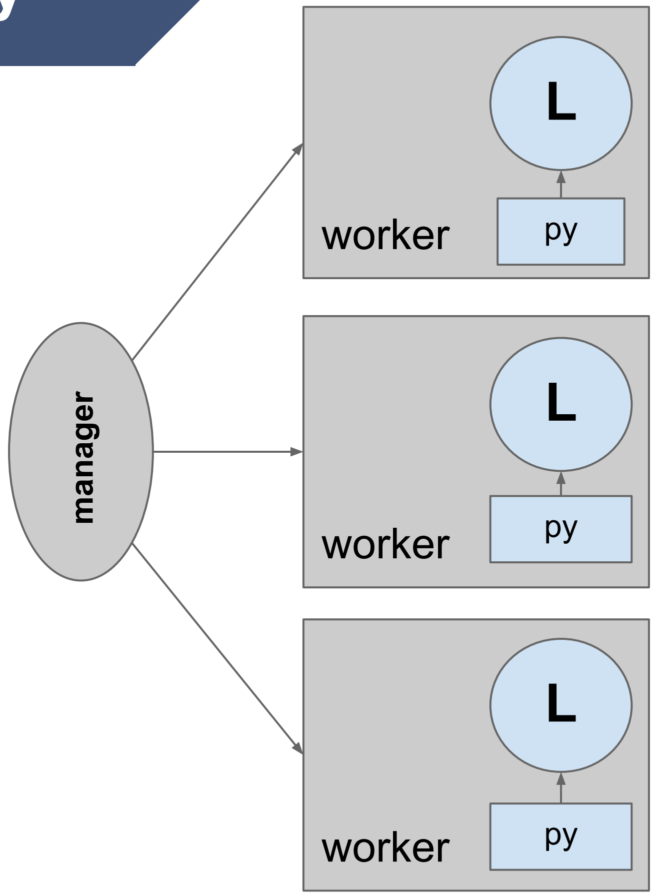
A Task runs to completion a single time, reading input files, and producing output files.



A LibraryTask contains a Function. It receives arguments, produces results, but then **stays running,** waiting for the next invocation.

Functions as a Service - Install Library

```
# Define ordinary Python functions
def my_sum(x, y):
    return x+y

def my_mul(x, y):
    return x*y

# Create a library object from functions
L = m.create_library_from_functions(
             "my_library",my_sum, my_mul)

# Install the library on all workers.
m.install_library(L)
```

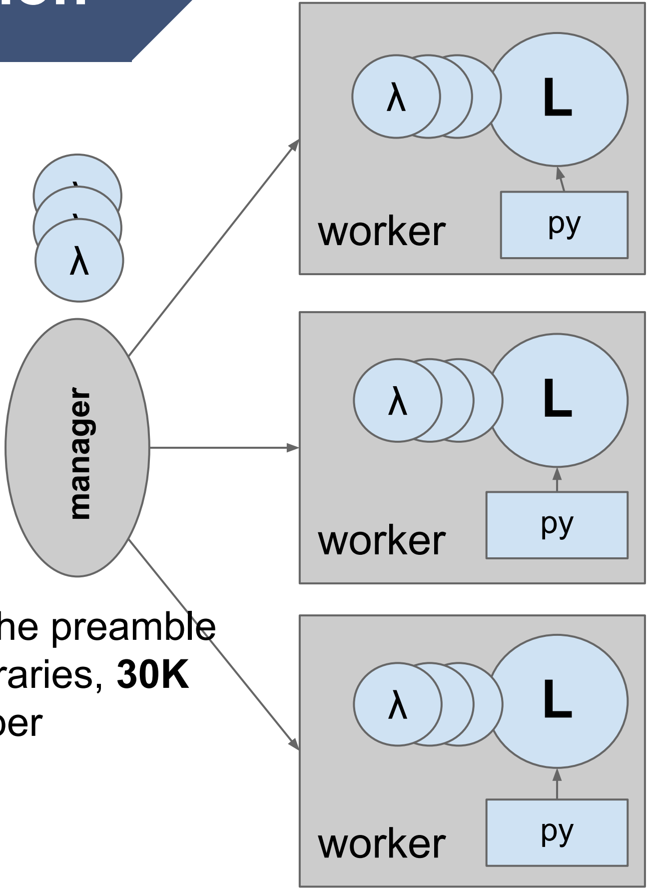


Functions as a Service - Invoke Function

```
# Define a function invocation and submit it

for i in range(1,100):
  t = vine.FunctionCall("my_library","my_sum",10,i)
```

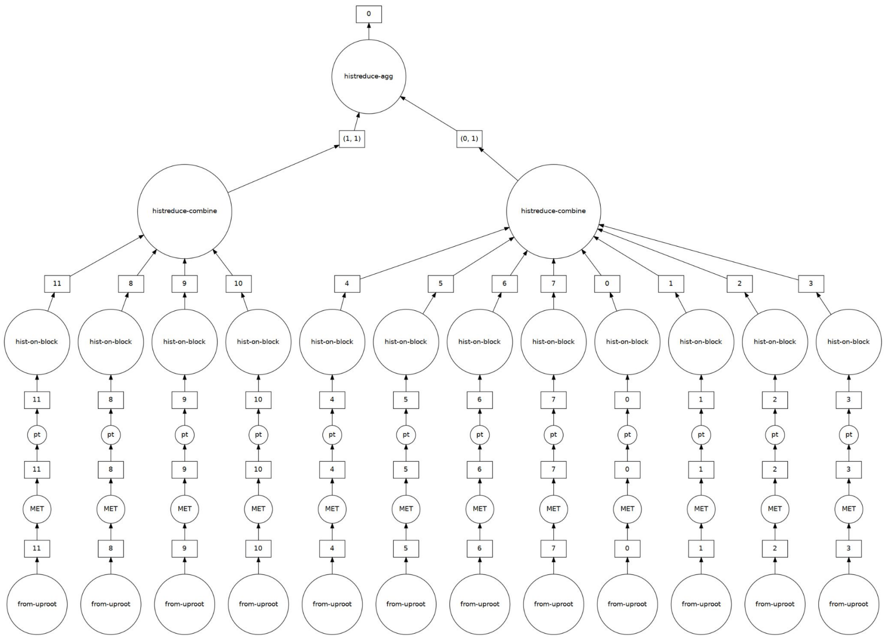
Simply converting "import tensorflow" into the preamble of a LibraryTask saves **1.2GB** of Python libraries, **30K** metadata system calls, and **5-10s** latency per FunctionCall.

David Simonetti and Thanh Phung



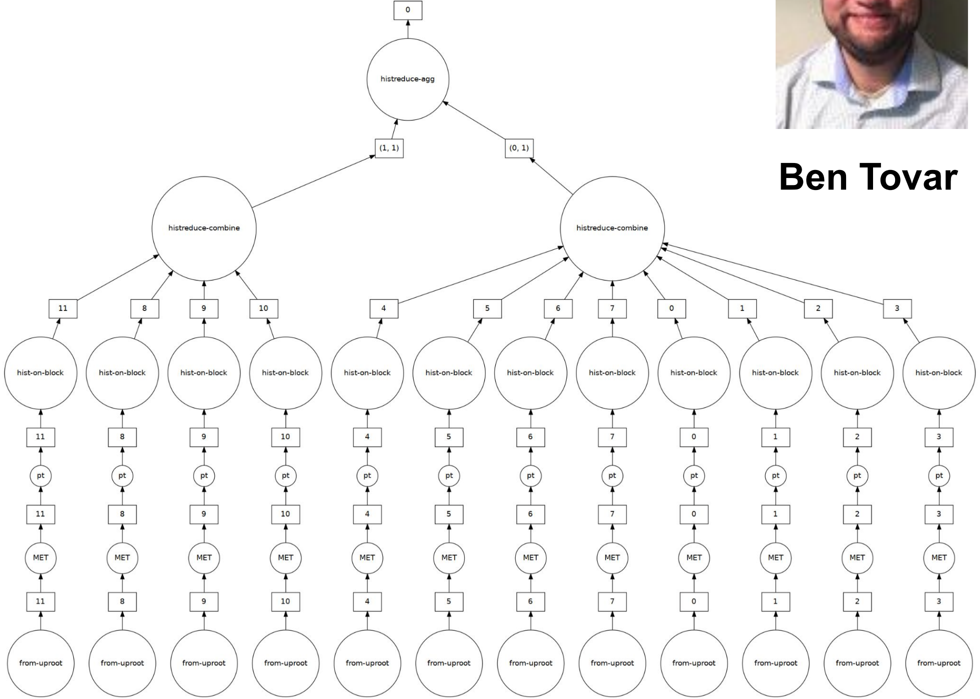
Building Up a Large DAG Manually

```
x = m.define_file()
y = m.define_file()
z = m.define_file()
. . .

a = Task()
b = FunctionCall()
c = FunctionCall()
. . .

a.add_input(x,"data")
a.add_output(y,"temp")
. . .
```



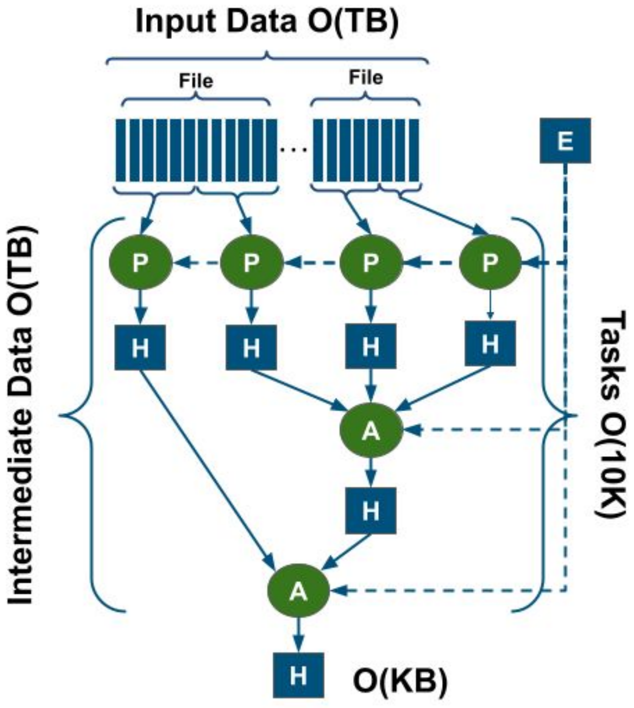
Building Up a Large DAG with Dask

```python
from ndcctools.taskvine import DaskVine
from coffea.nanoevents import NanoEventsFactory
import hist.dask as hda
import dask

dataset = get_dataset("SingleMu")
events = NanoEventsFactory.from_root(
    dataset,
    permit_dask=True,
    uproot_options={"chunks_per_file": 5}
    metadata={"dataset": "SingleMu"}
).events

hist = (
    hda.Hist.new.Reg(100, 0, 200, name ="met")
    .Double()
    .fill(events.MET.pt)
)

manager = DaskVine(name="my_manager")

hist.compute(
    scheduler=manager.get(),
    peer_transfers=True
    task_mode='function-calls'
    lib_resources={'cores':12, 'slots':12}
    import_modules=[numpy, scipy]
)
```



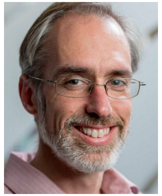
Ben Tovar

Reshaping HEP Data Analysis Apps



Example Application: DV3-Small

- Consumes 1.5TB Data
- Produces 17K Tasks
- Uses 2400 cores on 200 nodes.
- Runs in 3545s (~1 hour)



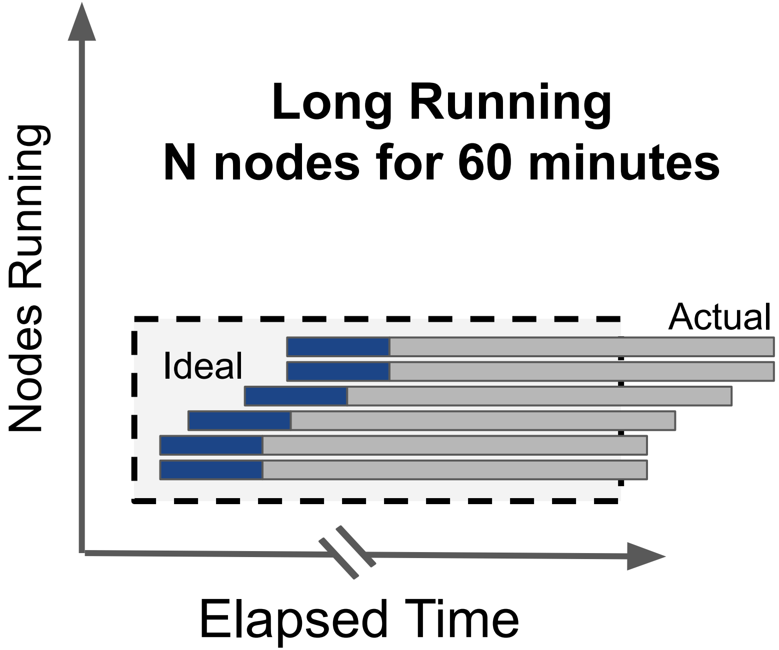
Kevin Lannon

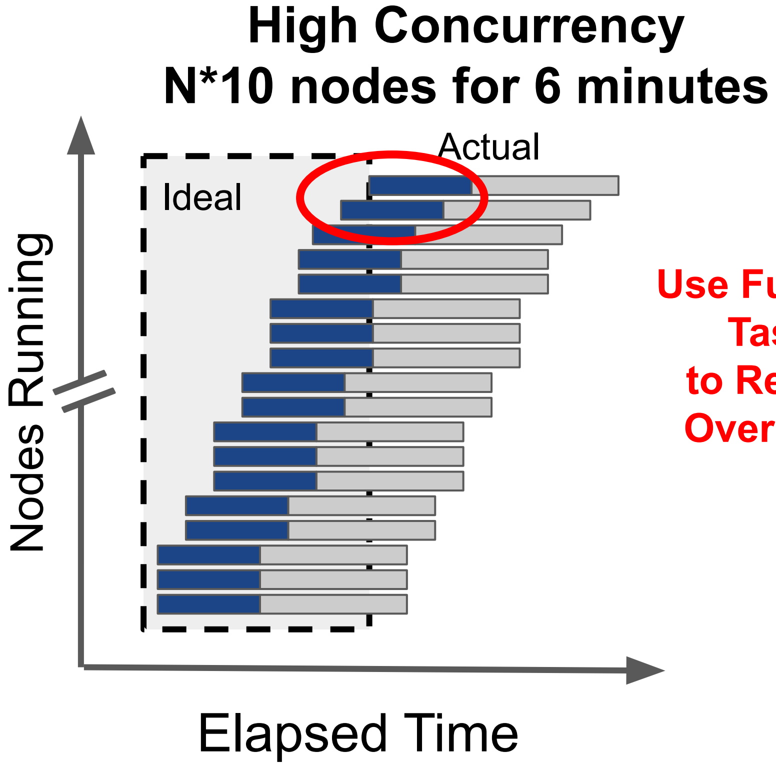
Kelci Mohrman

Connor Moore

This is not bad, but can we make it near-interactive?

More Tasks Requires Smaller Tasks and that Requires Lower Overhead!





Use Function Tasks to Reduce Overhead!

Evolution of Software Stack

Stack 1

DV3

Coffea

Dask

Work Queue

HDFS

3545s

Evolution of Software Stack

Stack 1	Stack 2
DV3	DV3
Coffea	Coffea
Dask	Dask
Work Queue	Work Queue
HDFS	VAST
3545s	3378s

Evolution of Software Stack

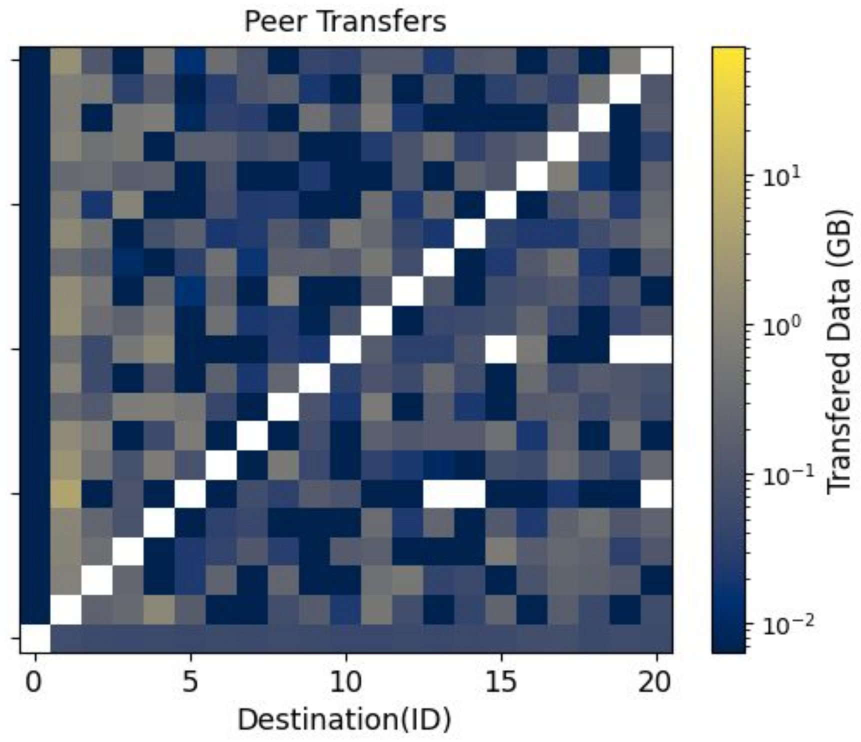
Stack 1	Stack 2	Stack 3
DV3	DV3	DV3
Coffea	Coffea	Coffea
Dask	Dask	Dask
Work Queue	Work Queue	TaskVine
HDFS	VAST	VAST
3545s	3378s	730s

Evolution of Software Stack

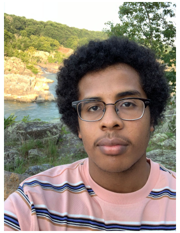
Stack 1	Stack 2	Stack 3	Stack 4
DV3	DV3	DV3	DV3
Coffea	Coffea	Coffea	Coffea
Dask	Dask	Dask	Dask
Work Queue	Work Queue	TaskVine	TaskVine + Functions
HDFS	VAST	VAST	VAST
3545s	3378s	730s	**272s**

13x faster!

Reason 1: Worker to Worker Data Access

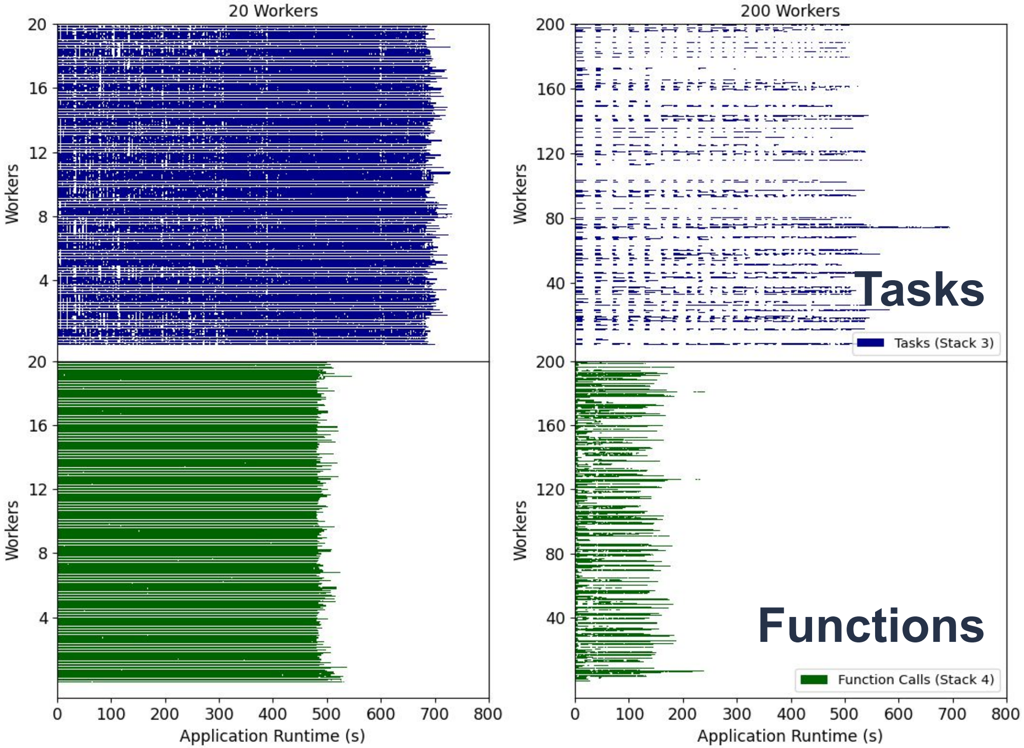


Both code and data are retained on worker local disks and transferred directly between workers as directed by the manager.

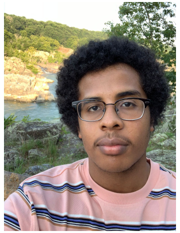


B. Sly-Delgado, J. Zhou, B.Tovar, and D.Thain, "Reshaping High Energy Physics Applications for Near-Interactive Execution Using TaskVine", to appear at Supercomputing 2024.

Reason 2: Functions are Lightweight!



Running LibraryTasks maintain code and data ready in memory so that function invocations are much lighter weight than standalone processes.



B. Sly-Delgado, J. Zhou, B.Tovar, and D.Thain, "Reshaping High Energy Physics Applications for Near-Interactive Execution Using TaskVine", to appear at Supercomputing 2024.

High Throughput Analysis Stack

CCTools

dask — Familiar Python interface to code and data.

TaskVine — Lightweight task and data scheduling.

HTCondor — High throughput resource management.

High Throughput Analysis Stack

CCTools

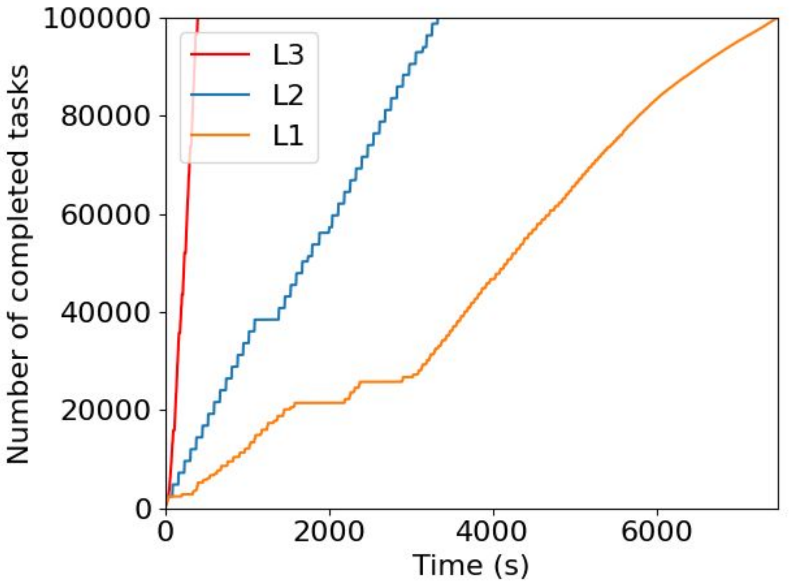
Parsl — Python based workflow generation.

TaskVine — Lightweight task and data scheduling.

HTCondor — High throughput resource management.

Parsl + TaskVine Exploiting Functions

Large Scale Neural Network Inference
100K tasks on 150 x 32c workers



- L1 - Traditional Access to HPC Filesystem.
- L2 - Tasks with data cached on workers.
- **L3 - Functions retaining state at workers.**




T. Phung, C. Thomas, L. Ward, K. Chard, D. Thain, **Accelerating Function-Centric Applications by Discovering, Distributing, and Retaining Reusable Context in Workflow Systems**, *ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, June, 2024.

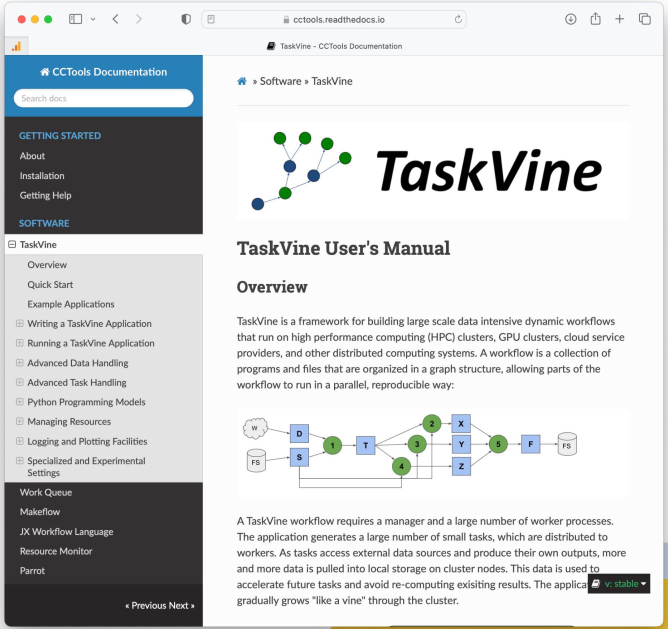
Current Status of TaskVine

This work was supported by NSF Award OAC-1931348

- **TaskVine** is a component of the Cooperative Computing Tools (cctools) from Notre Dame alongside Makeflow, Work Queue, Resource Monitor, etc.
- Release 7.11.1 made in June 2024.
- Research software with an engineering process: issues, tests, manual, examples.
- We are eager to collaborate with new users on applications and challenges!

conda install -c conda-forge ndcctools

https://cctools.readthedocs.io

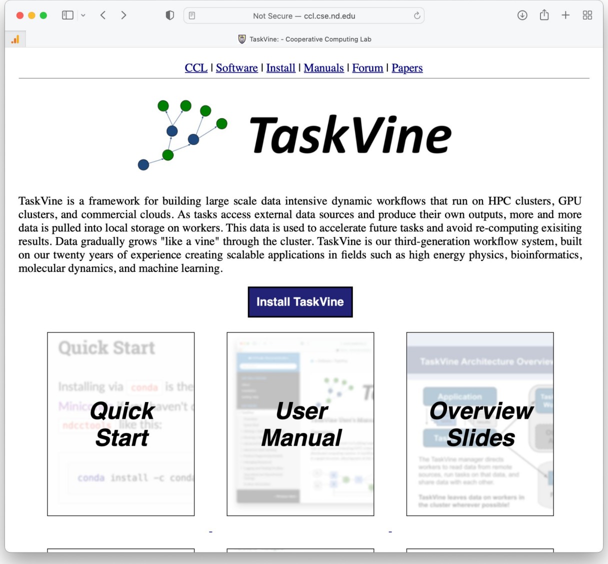


For more information…

This work was supported by NSF Award OAC-1931348

https://cctools.readthedocs.io
https://dthain.github.io

Cooperative Computing Lab
Staff and Students




```
conda install -c conda-forge ndcctools
```