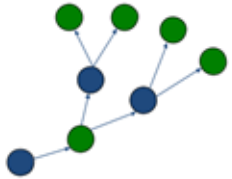
UNIVERSITY OF NOTRE DAME

Wrangling Massive Task Graphs with Vine Reduce

Ben Tovar and Douglas Thain
University of Notre Dame

Throughput Computing
University of Wisconsin
11 June 2026

TaskVine



Wrangling Massive Task Graphs

CCTools

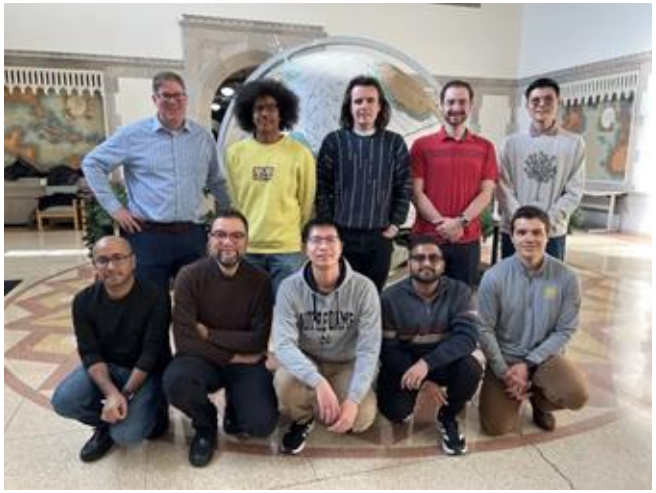
- From Workflows to Task Graphs
- Task Graphs with TaskVine
- But What Happens at Large Scale?
- Hierarchical Reduction with Vine Reduce
- Lessons Learned and What's Next?

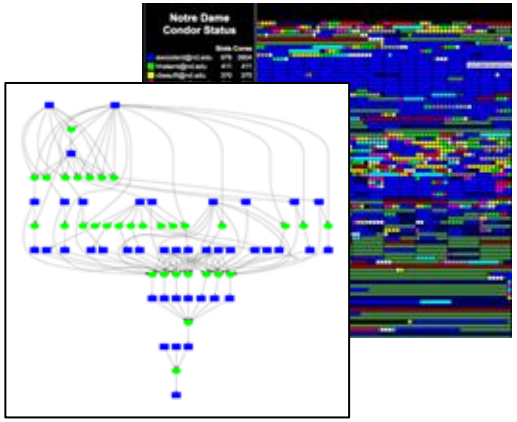
The Cooperative Computing Lab

CCTools

UNIVERSITY OF NOTRE DAME

http://ccl.cse.nd.edu





Makeflow

TaskVine

Work Queue

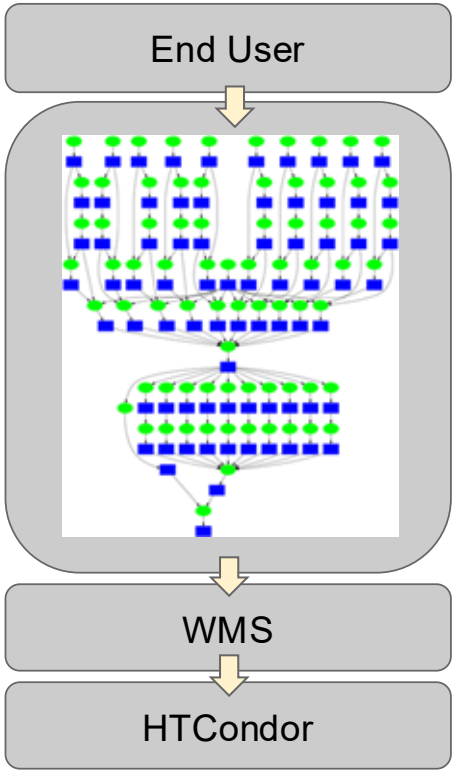
floability

NASA

DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA

NSF

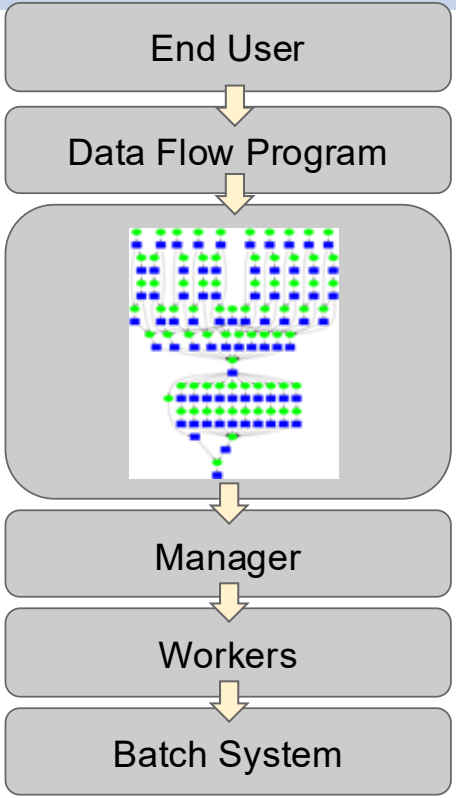
Everyone Knows About Workflows



Conventional Workflow Systems

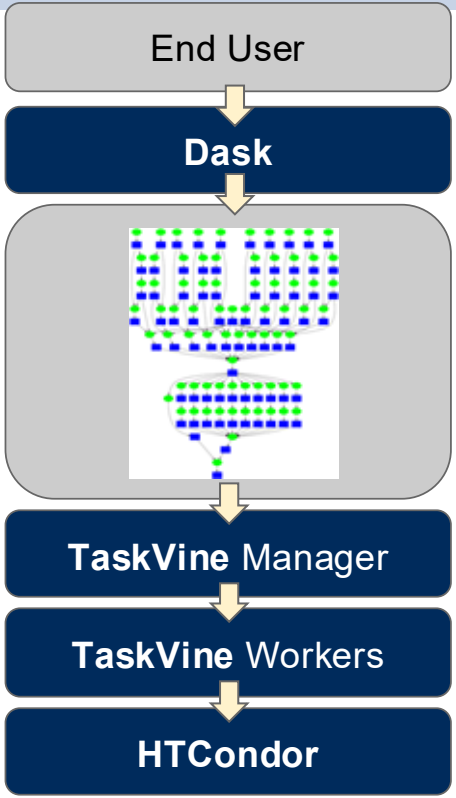
- Each green square an executable program to be run via Unix fork/exec. (mins to hours)
- Each blue square a file in the filesystem accessed via Unix. (MB to GB)
- End user writes out the individual nodes and sees the entire graph.
- A workflow manager puts the jobs in order and submits them to HTCondor.

Task Graphs are a Variation on Workflows



- End user writes a **dataflow program** in a familiar programming language.
- Each green circle a **function** to be **called** with **arguments**. (sec-min)
- Each blue square a data item to be **accessed in memory** by a function. (KB-GB)
- The graph is generated automatically, and the user (often) **does not view** or manipulate it.
- Manager dispatches tasks to workers.
- Workers are deployed on a batch system.

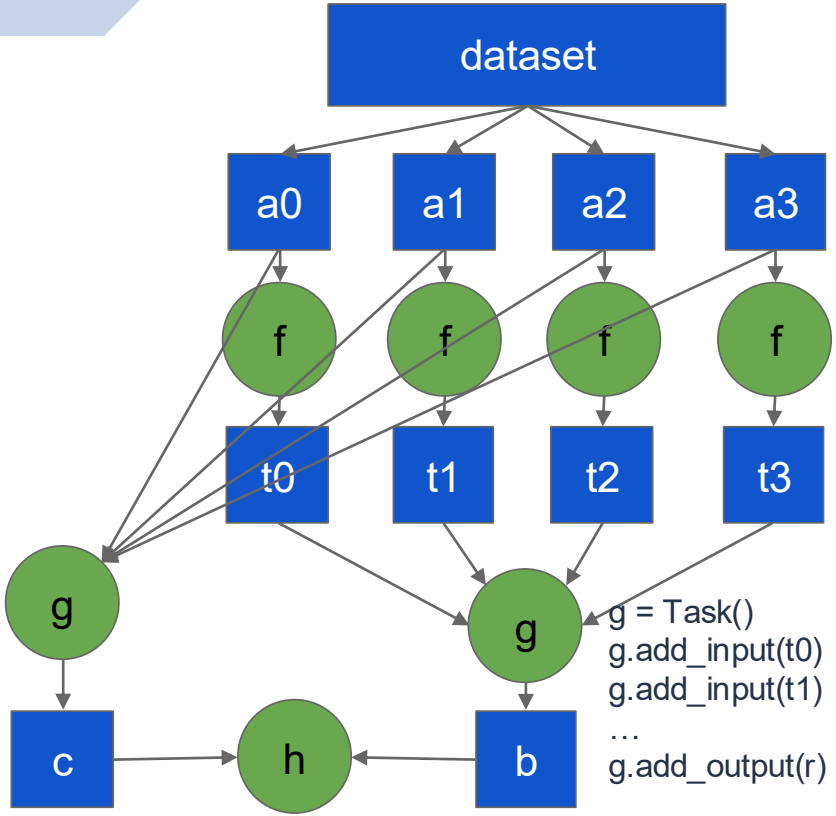
Task Graphs are a Variation on Workflows



- End user writes a **dataflow program** in a familiar programming language.
- Each green circle a **function** to be **called** with **arguments**. (sec-min)
- Each blue square a data item to be **accessed in memory** by a function. (KB-GB)
- The graph is generated automatically, and the user (often) **does not view** or manipulate it.
- Manager dispatches tasks to workers.
- Workers are deployed on a batch system.

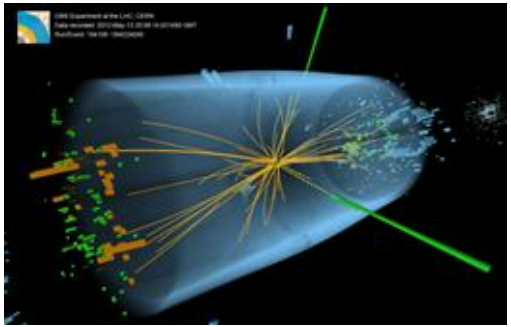
Dataflow Programming in General

```
a = dataset("/path/…").split(n)
b = a.map(f).reduce(g)
c = a.reduce(g)
d = h(a,b)
# nothing happens until:
print (d)
```



Example: CMS Experiment at the LHC

- The Compact Muon Solenoid (CMS) is one of four experiments at the Large Hadron Collider (LHC). The LHC generates proton beams that collide and generate new subatomic particles in order to learn new physics interactions and discover new particles. (e.g. Higgs Boson)
- CMS observes 30M collisions ("events") per second, resulting in several PB of useful data each year. Physicists must write analysis code to sort through these events, select "interesting" categories, compute derived properties, and summarize new physical interactions.
- Rapid iterative development in Python using well known numeric toolkits (numpy, pandas, etc).
- Potential to execute with extreme parallelism on clusters with thousands of nodes!



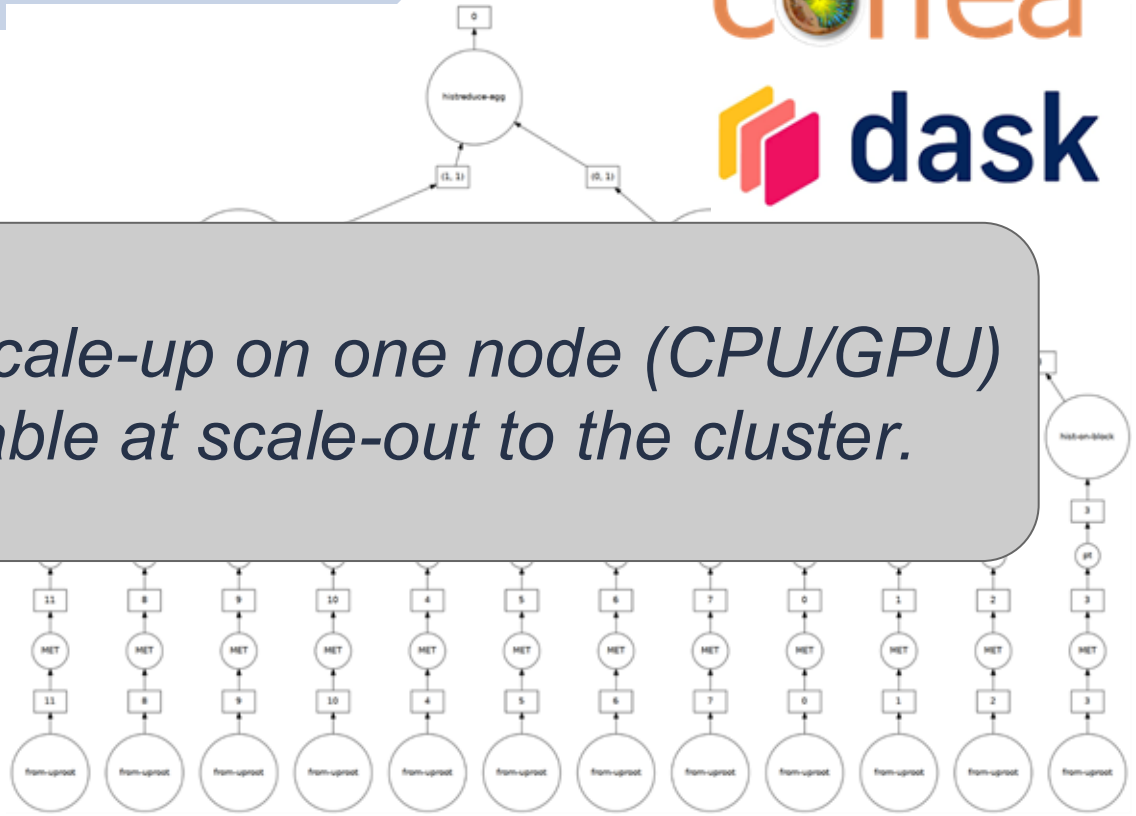
CMS Data Analysis Apps with Dask

CCTools

coffea

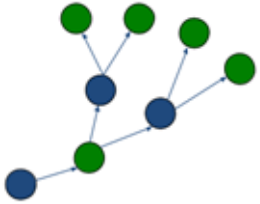
dask

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

dataset = get_dataset("SingleMu")

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]
)
```



Dask is excellent at scale-up on one node (CPU/GPU)
…. but not very stable at scale-out to the cluster.

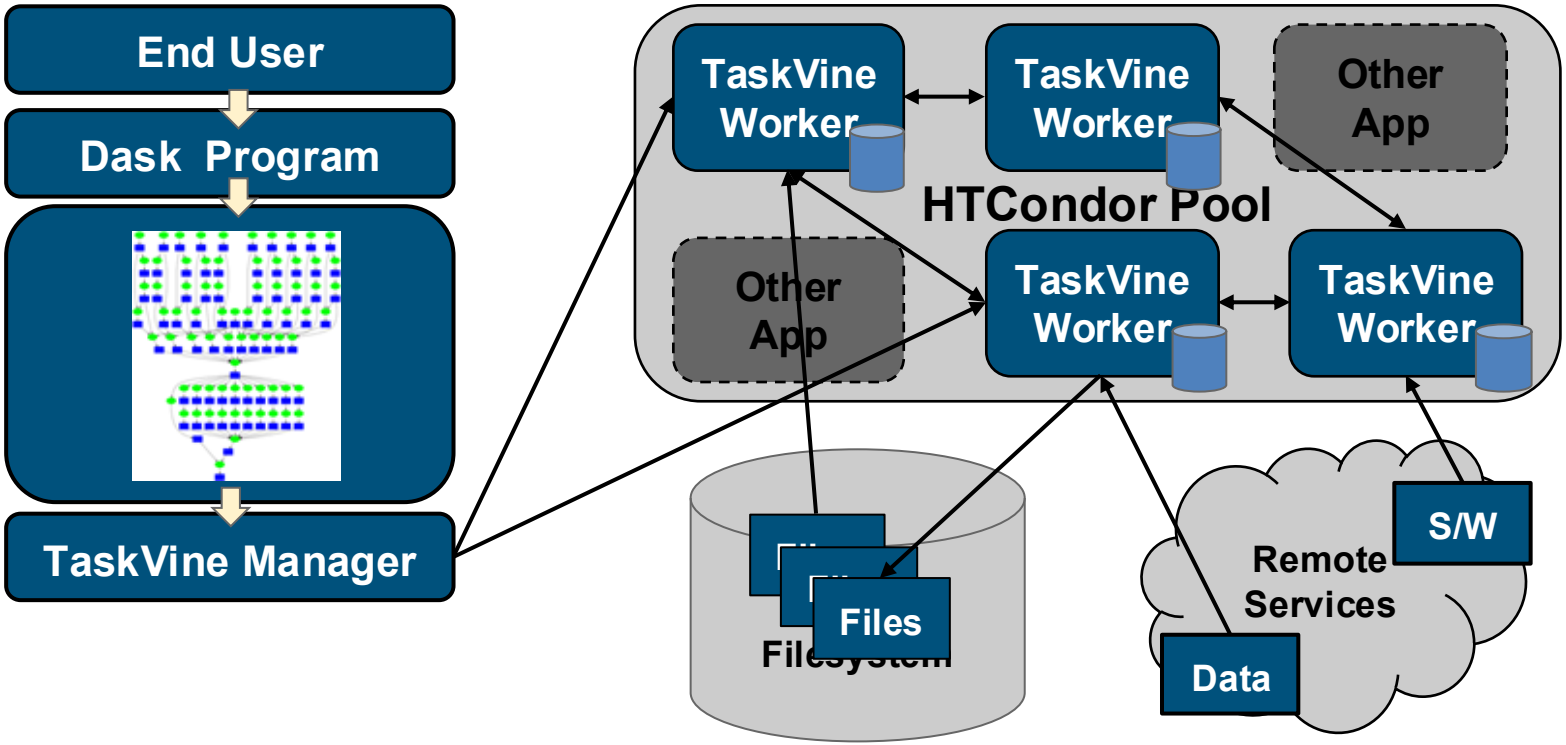
TaskVine

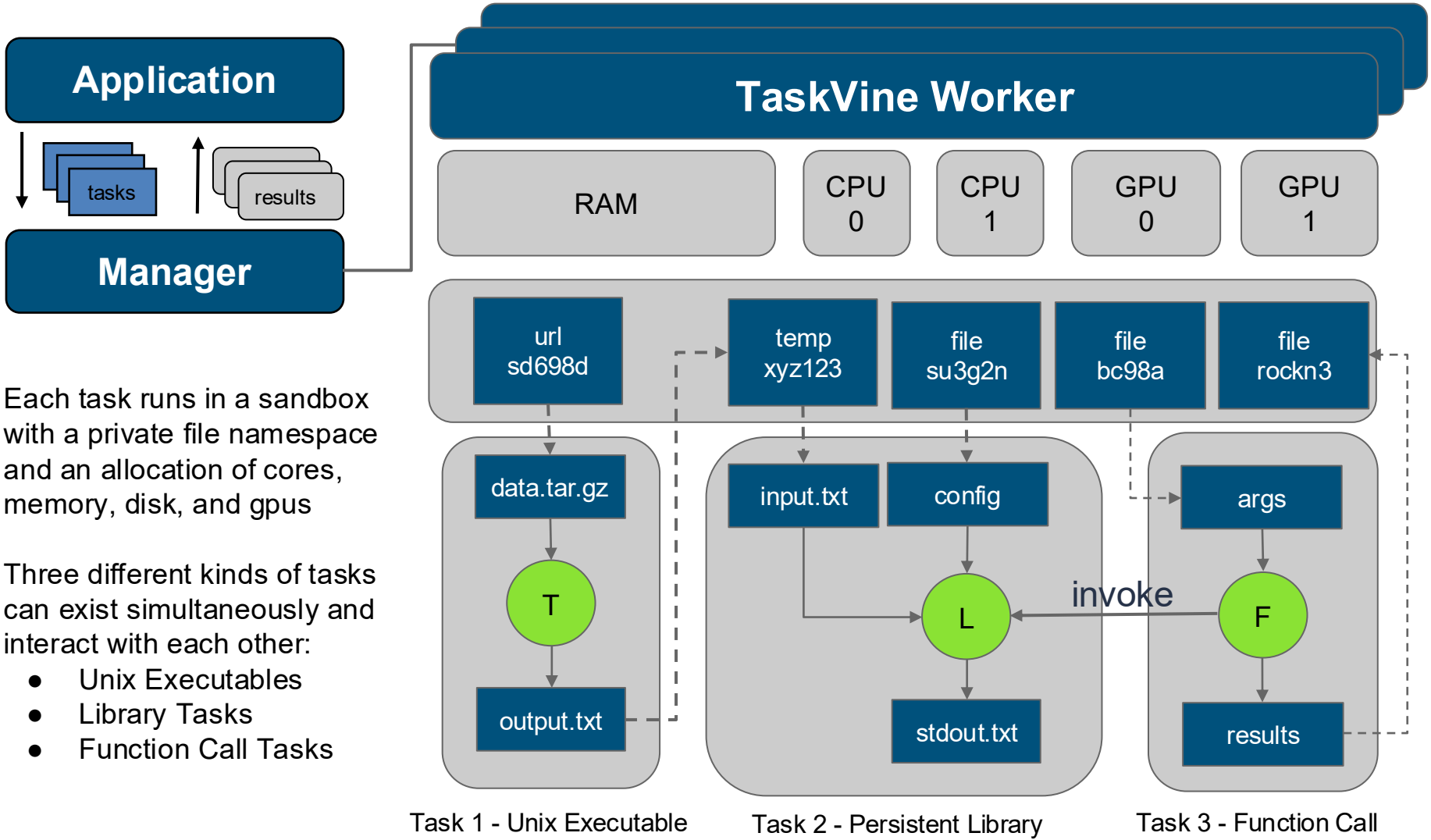


TaskVine is a scheduler for data intensive, high throughput, large scale, task graphs on opportunistic resources:

- Slice up cores/memory/gpus arbitrarily among tasks.
- Automatically choose task resource allocations.
- Intermix Unix tasks and FaaS ("serverless") tasks.
- Maintain data on the workers, schedule for locality.
- Handle failures/retry of workers, tasks, transfers, etc.
- Integrates effortlessly with HTCondor as a resource provider.

Dask + TaskVine Architecture



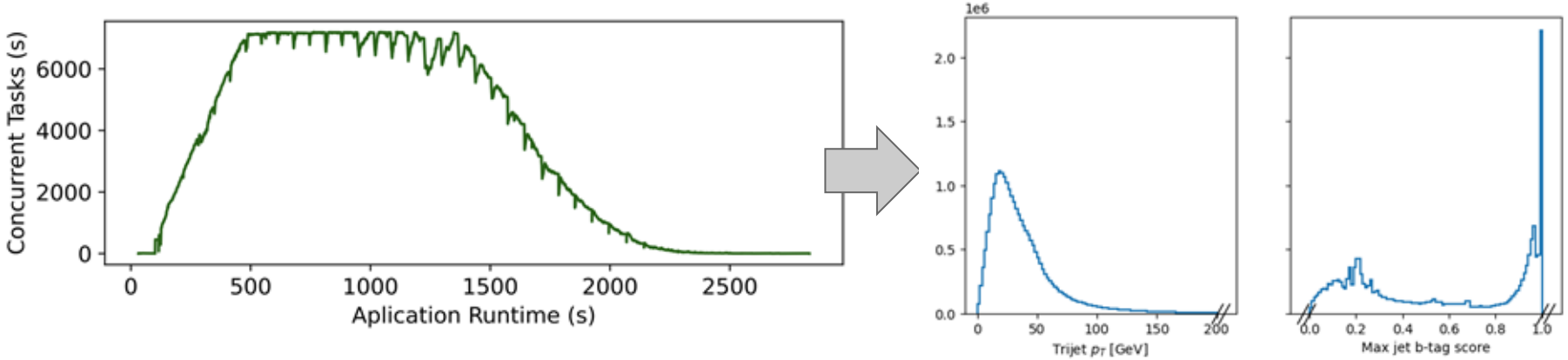


Each task runs in a sandbox with a private file namespace and an allocation of cores, memory, disk, and gpus

Three different kinds of tasks can exist simultaneously and interact with each other:

- Unix Executables
- Library Tasks
- Function Call Tasks

Task Graph Execution with TaskVine



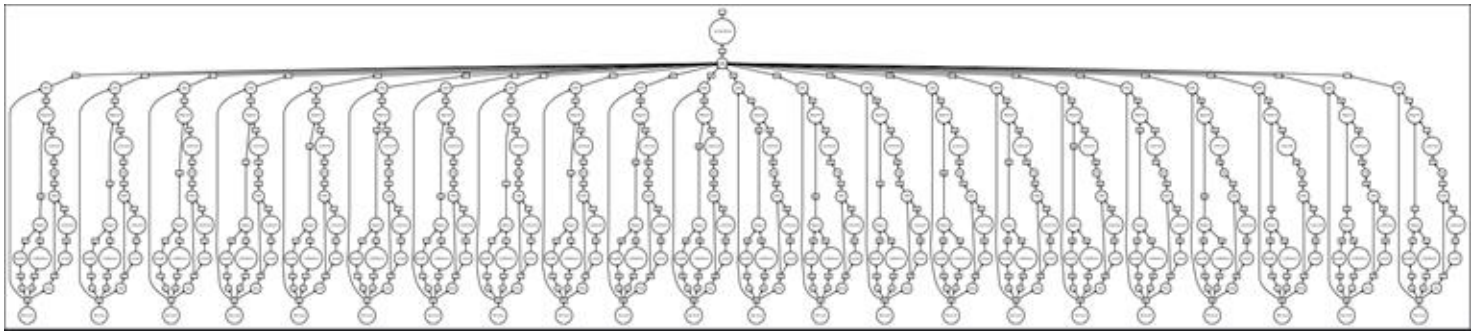
DV5 data analysis program generated task graph of 185,000 tasks, running on 7000 cores done in 41 minutes.

Barry Sly-Delgado, Ben Tovar, Jin Zhou, and Douglas Thain,
Reshaping High Energy Physics Applications for Near-Interactive Execution Using TaskVine,
ACM/IEEE Supercomputing, pages 1-11, November, 2024. DOI: 10.1109/SC41406.2024.00068

Let's go even larger!

(multiply this graph by 419X to get the full task graph.)

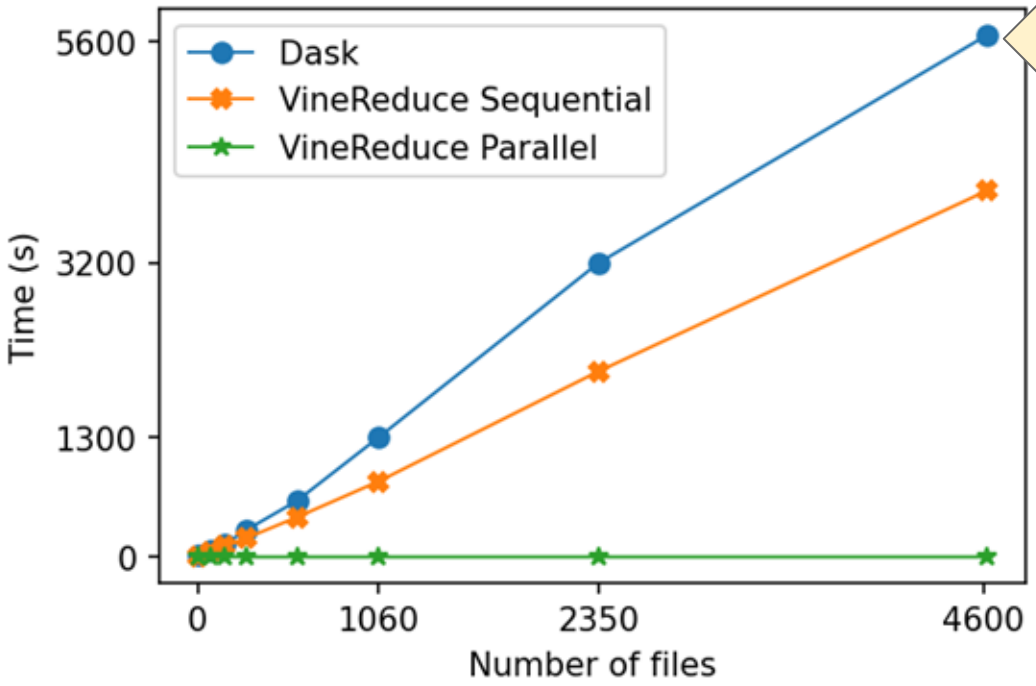
Kelci Mohrman (UFL)



- Cortado data analysis application:
  419 datasets, 19631 files, 14TB, 12 billion events

- **Generating the complete Dask graph the simple way takes 20 hours even before execution starts!**

Problem: Setup Time is Atrocious!

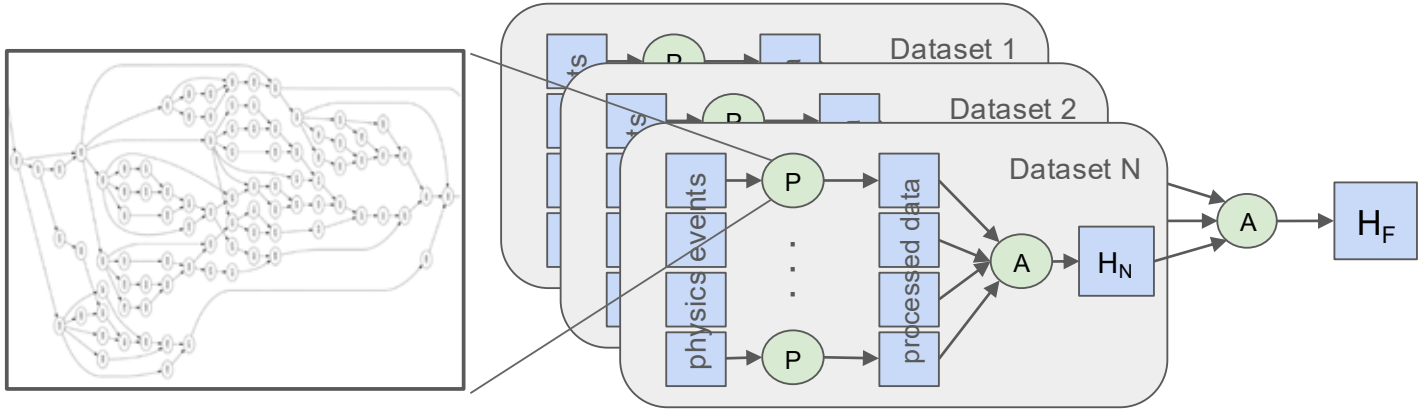


Just **creating** the graph starts to rival the distributed execution time!

***(Maybe** we can optimize this by a constant factor, but it's still going to be large.)*

(Ignore this line for now.)

Insight: There is a Hierarchical Structure!



This task graph is actually a Map-Reduce.
… of Map-Reduces
… of O(1000) node task graphs.

VineReduce API

```
def source_postprocess(chunk_info, **source_args):
    from coffea.nanoevents import NanoEventsFactory, PFNanoAODSchema
    import math
    import os

    cores = source_args.get("cores", int(os.environ.get("CORES", 1)))
    num_entries = chunk_info["entry_stop"] - chunk_info["entry_start"]
    step = source_args.get("chunk_step_size", math.ceil(num_entries / cores))

    steps = []
    start = chunk_info["entry_start"]

    while start < chunk_info["entry_stop"]:
        end = min(start + step, chunk_info["entry_stop"])
        steps.append([start, end])
        start = end

    d = {
        chunk_info["file"]: {
            "object_path": chunk_info["object_path"],
            "metadata": chunk_info["metadata"],
            "steps": steps,
        }
    }

    events = NanoEventsFactory.from_root(
        d,
        schemaclass=PFNanoAODSchema,
        uproot_options={"timeout": 300},
        metadata=dict(chunk_info["metadata"]),
    )

    return events.events()
```

```
def analysis(events):
    import awkward as ak
    import fastjet
    import scipy

    dataset = events.metadata["dataset"]
    print(dataset)

    events["PFCands", "pt"] = events.PFCands.pt * events.PFCands.puppiWeight

    cut_to_fix_softdrop = ak.num(events.FatJet.constituents.pf, axis=2) > 0
    events = events[ak.all(cut_to_fix_softdrop, axis=1)]

    trigger = ak.zeros_like(ak.firsts(events.FatJet.pt), dtype="bool")
    for t in triggers["2017"]:
        if t in events.HLT.fields:
            trigger = trigger | events.HLT[t]
    trigger = ak.fill_none(trigger, False)

    events["FatJet", "num_fatjets"] = ak.num(events.FatJet)

    goodmuon = (
        (events.Muon.pt > 10)
        & (abs(events.Muon.eta) < 2.4)
        & (events.Muon.pfRelIso04_all < 0.25)
        & events.Muon.looseId
    )

    nmuons = ak.sum(goodmuon, axis=1)

    goodelectron = (
        (events.Electron.pt > 10)
        & (abs(events.Electron.eta) < 2.5)
        & (events.Electron.cutBased >= 2)  # events.Electron.LOOSE
    )
    nelectrons = ak.sum(goodelectron, axis=1)

    ntaus = ak.sum(
        (
            (events.Tau.pt > 20)
```

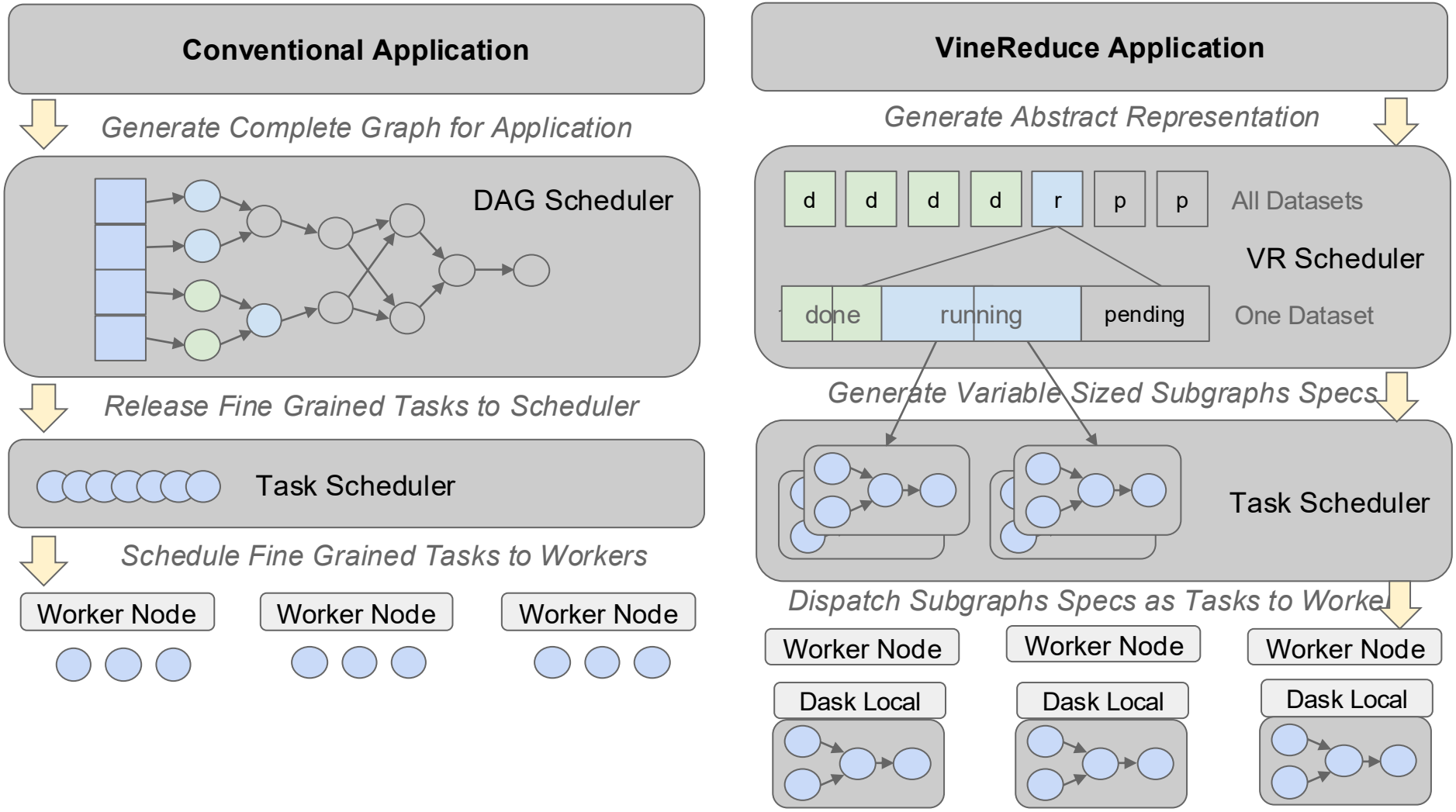
```
def
def
def
def
vr
```

preprocess, postprocess,
processor, reducer)

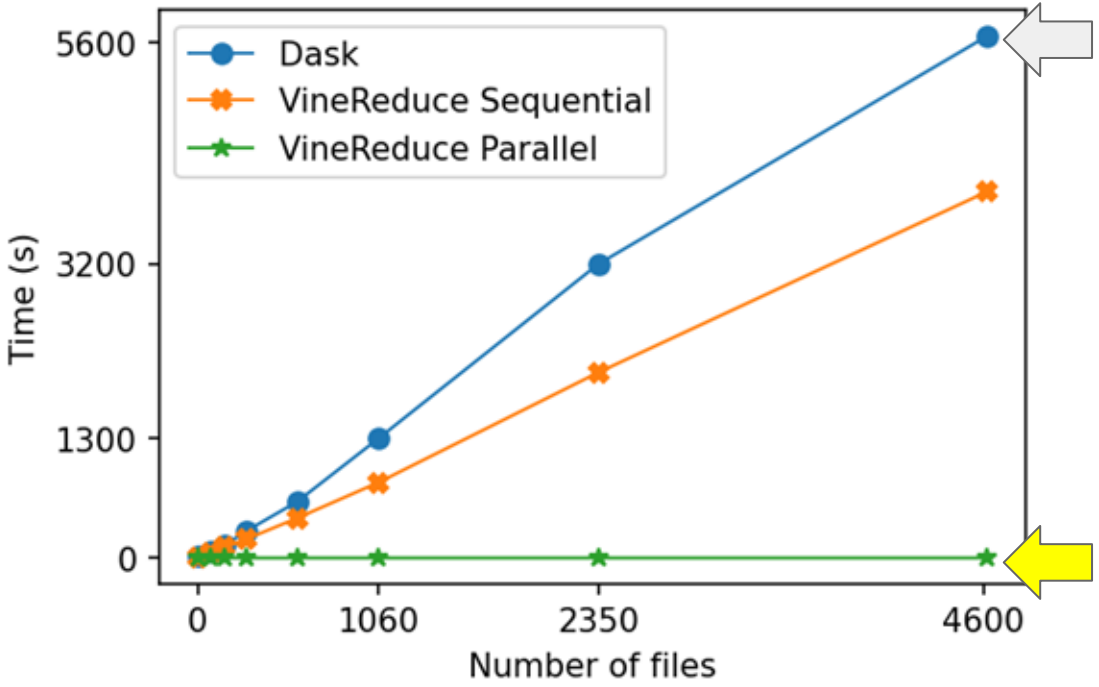
ta.

as

on the


Conventional Application
Generate Complete Graph for Application
DAG Scheduler
Release Fine Grained Tasks to Scheduler
Task Scheduler
Schedule Fine Grained Tasks to Workers
Worker Node
Worker Node
Worker Node
VineReduce Application
Generate Abstract Representation
d
d
d
d
r
p
p
All Datasets
VR Scheduler
done
running
pending
One Dataset
Generate Variable Sized Subgraphs Specs
Task Scheduler
Dispatch Subgraphs Specs as Tasks to Worker
Worker Node
Worker Node
Worker Node
Dask Local
Dask Local
Dask Local

Advantage 1: Setup Time Disappears

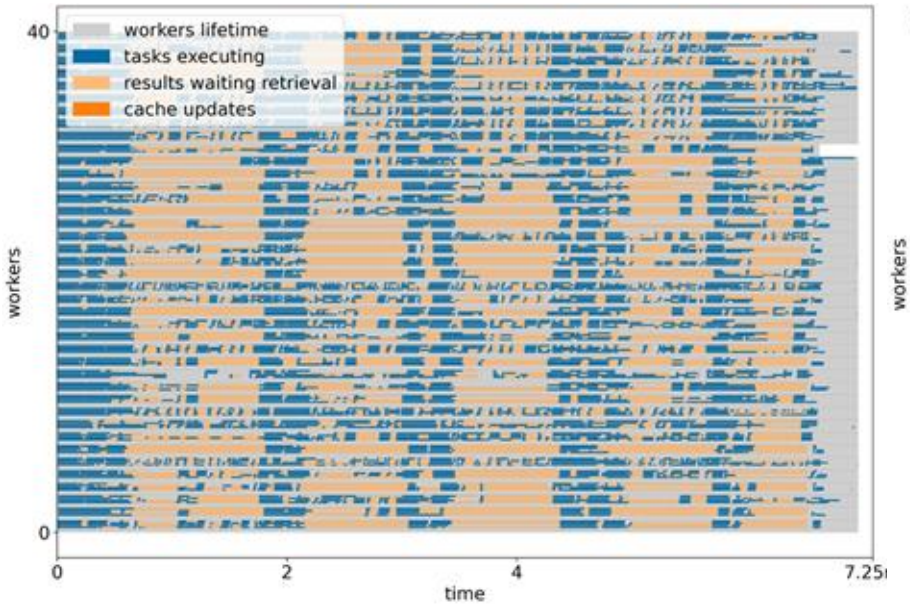


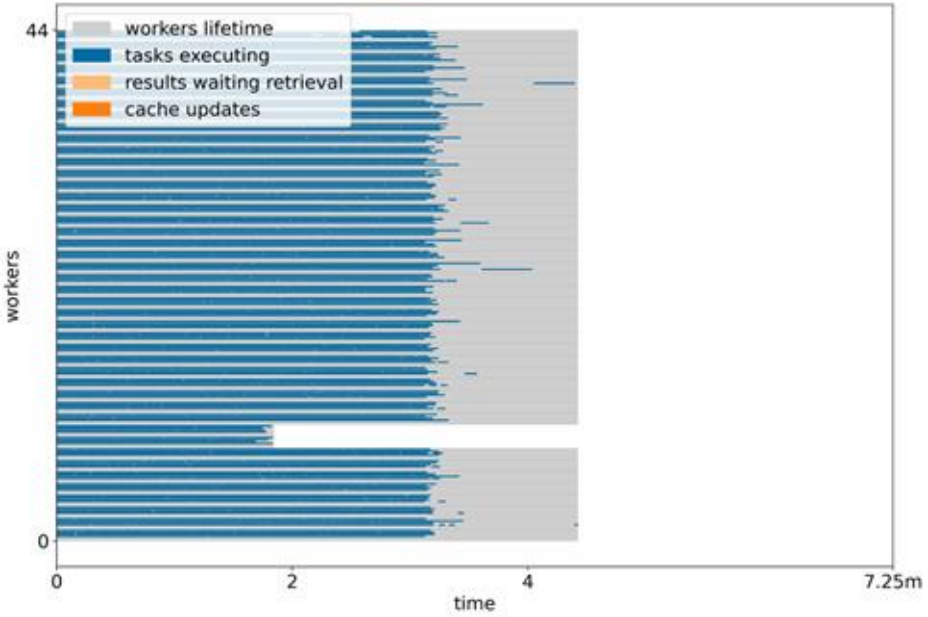
Just **creating** the graph starts to rival the distributed execution time!

*(**Maybe** we can optimize this by a constant factor, but it's still going to be large.)*

(most) graph construction delegated to the workers!

Advantage 2: Workers Tightly Packed

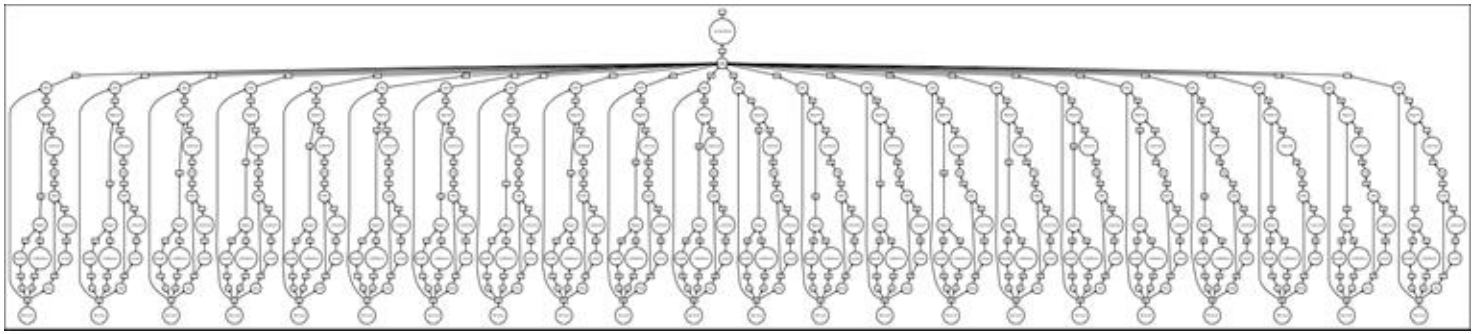




Impact on Cortado Data Analysis App:

(multiply this graph by 419X to get the full task graph.)

Kelci Mohrman (UFL)



- Cortado data analysis application:
  419 datasets, 19631 files, 14TB, 12 billion events
- ~~Generating the complete Dask graph the simple way takes **20 hours** even before execution starts!~~
- VineReduce with Cortado runs from setup to completion on on 250 8-core nodes (1600 cores) in **5.5 hours**.

Ruminations

- Important to match hardware and software hierarchies.
  - Was the software designed with the HW in mind?
- VineReduce is a static framework.
  - Can we identify dispatchable subdags dynamicaly?
- Programming Language Theory:
  - Eager/lazy evaluation, generators, functionals…
- Distance Between Intent and Program is Increasing:
  - Prompt -> LLM -> FP -> DAG -> Tasks -> Workers

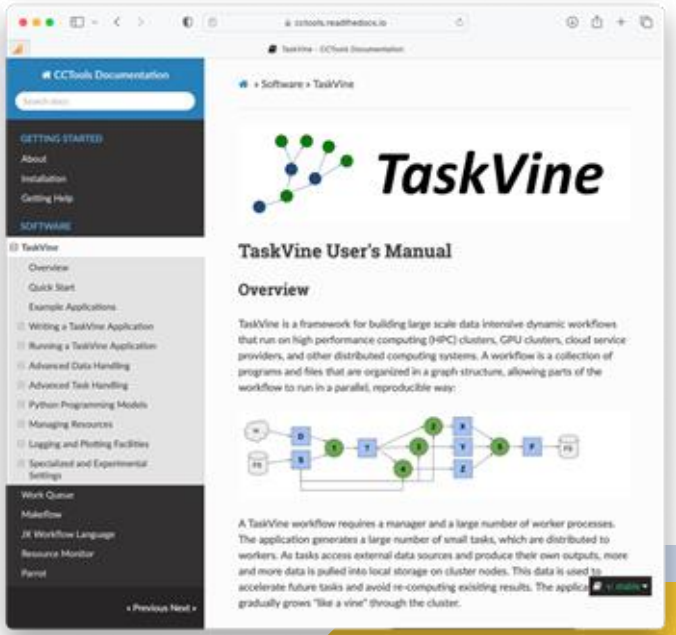
Give TaskVine a Try!

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 7.17 made in May 2026.
- Research software with an engineering process: issues, tests, manual, examples.
- We are eager to collaborate with new users on applications and challenges!

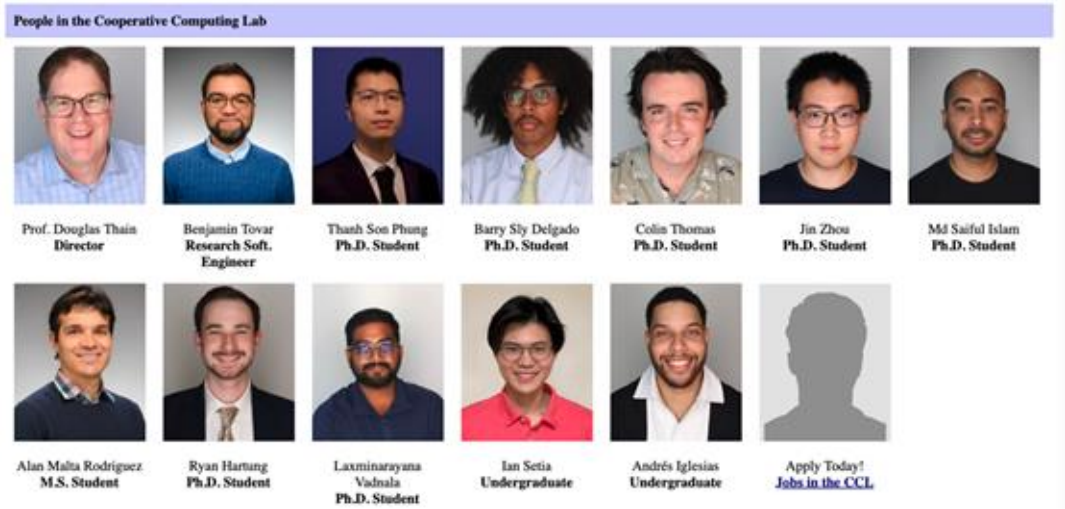
conda install -c conda-forge ndcctools

https://cctools.readthedocs.io

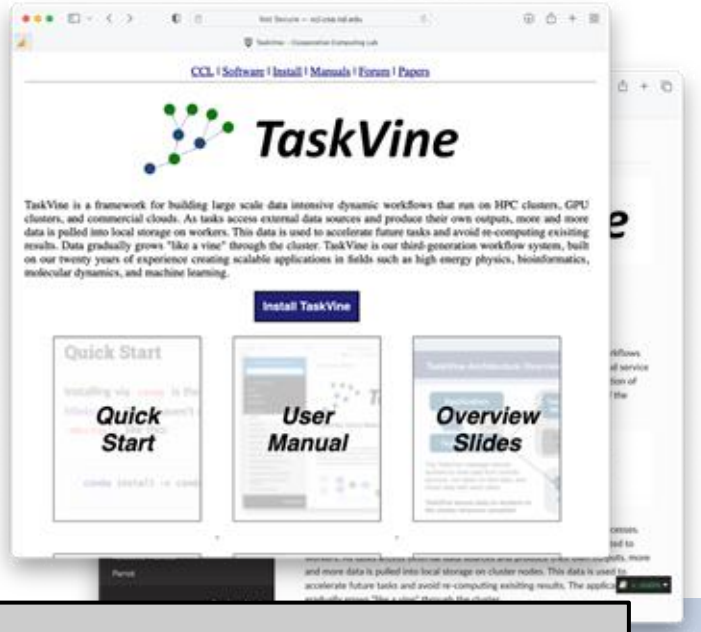


For more information…

This work was supported by
NSF Award OAC-1931348



CCL Staff and Students



https://cctools.readthedocs.io
https://ccl.cse.nd.edu

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