UNIVERSITY OF NOTRE DAME

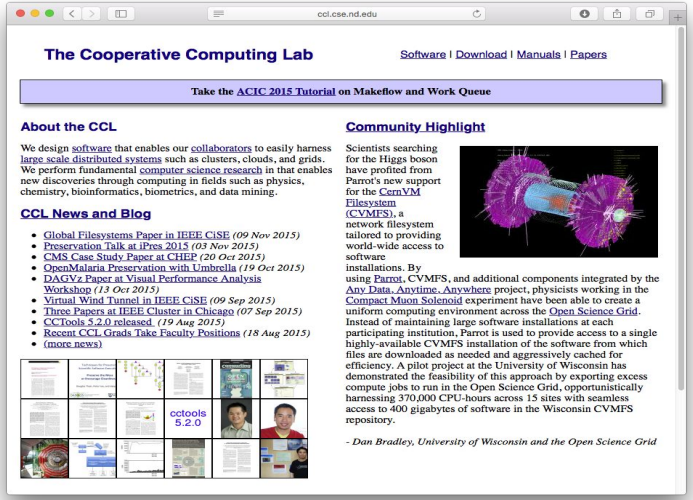
Data Intensive Computing with TaskVine

Douglas Thain and the CCL Team
University of Notre Dame
Greater Chicago Area
Systems Research Workshop, April 2023

CCTools

The Cooperative Computing Lab

CCTools



http://ccl.cse.nd.edu

We *collaborate with people* who have large scale computing problems in science, engineering, and other fields.

We *operate computer systems* on the O(10,000) cores: clusters, clouds, grids.

We *conduct computer science* research in the context of real people and problems.

We *develop open source software* for large scale distributed computing.

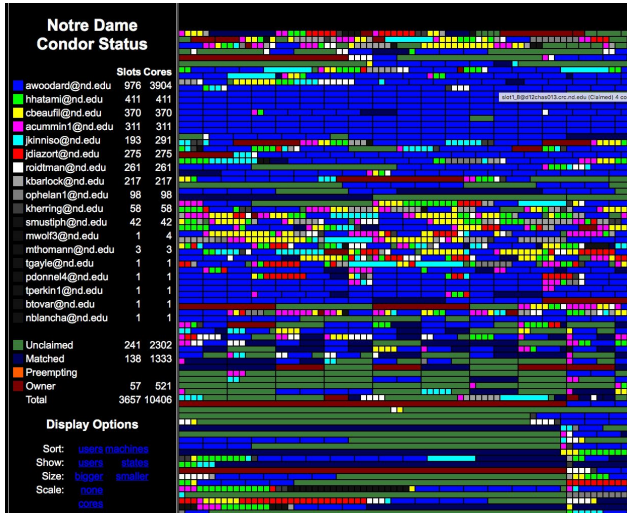
Scientific computing usually starts here:

python™

CONDA

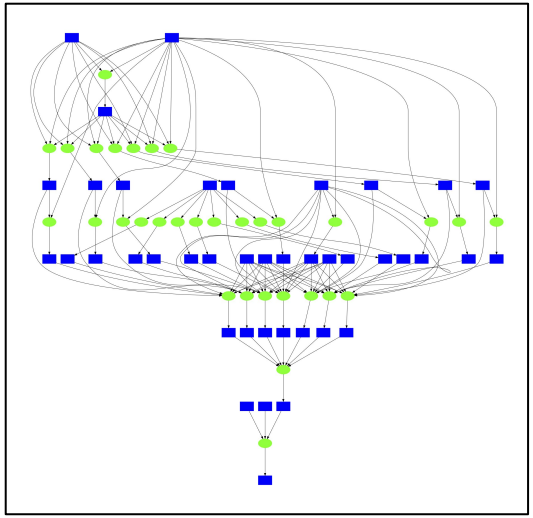
But how do you scale up to clusters?

Computing Facility



Workflows as a Computational Abstraction

A workflow is a collection of existing programs (functions) along with files (data objects) joined together into a large graph expressing dependencies. Allows for parallelism, distribution, and provenance without rewriting everything from scratch.

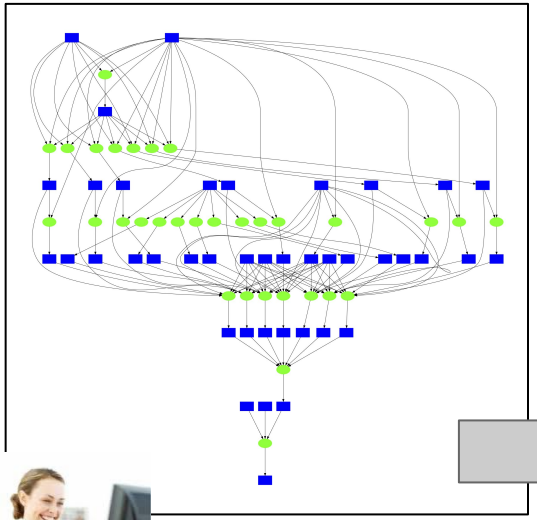


Research and Design Problems:

- Resource Allocation
- Scaling and Performance
- Data Management
- Reliability
- Portability
- Reproducibility

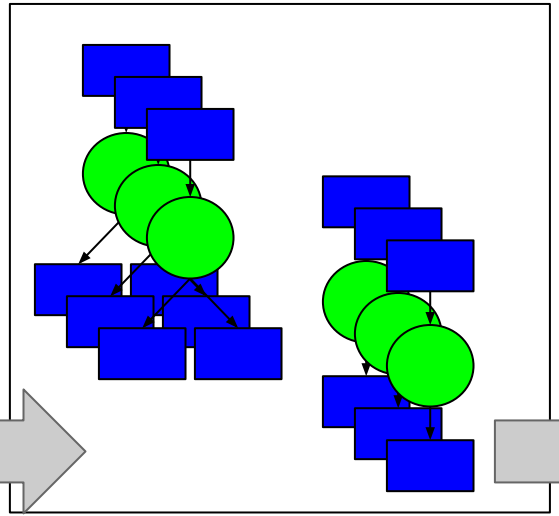
Workflow Management Systems

Workflow Manager



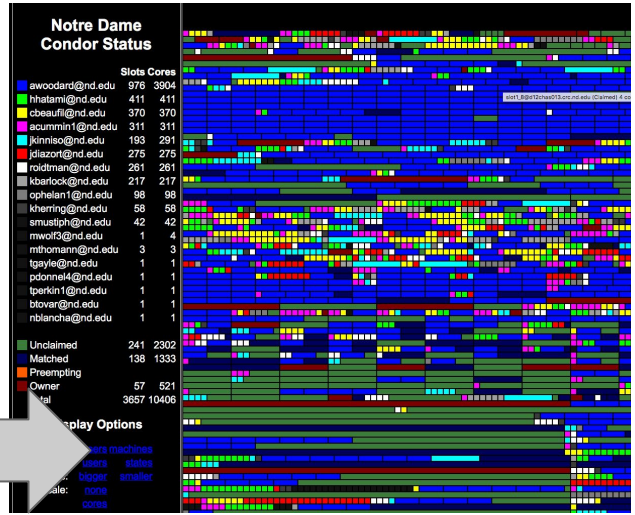
Express overall workflow structure, components, constraints, and goals.

Task / Data Scheduler



Assign ready tasks and data objects to resources in the cluster, subject to runtime constraints.

Computing Facility



Execute tasks on computational resources, store and move data between nodes.

http://workflows.community

workflows community initiative

Resources ▾ Research ▾ Events ▾ Members Jobs About **Get involved!**

Zenodo, 2023

Workflows Community Summit

National Academies of Sciences, Engineering, and Medicine, 2022

Zenodo, 2022

Workflows Community Summit

Workflows Community Initiative Retweeted

eScience 2023 @escie... · Apr 19

We have another insightful workshop called ReWorDS that discusses the #reproducibility, #data management, and #security efforts of #eScience, #HPC, and #AI workflows. Check more: sites.google.com/vols.utk.edu/r... @paulaolaya22 @JayLofstead #sciences #workflows

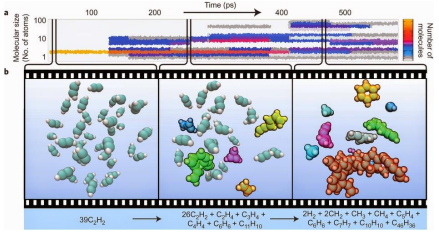
sites.google.... rewords23

9

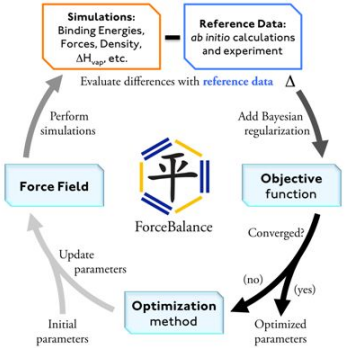
Workflows Community

Some Workflow Applications

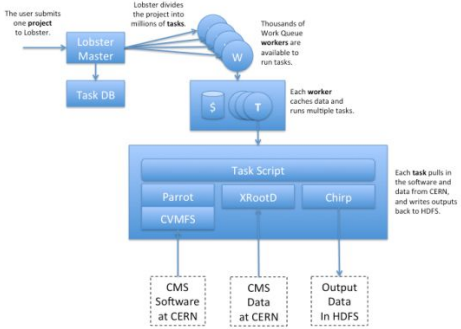
Nanoreactors
ab-initio Chemistry



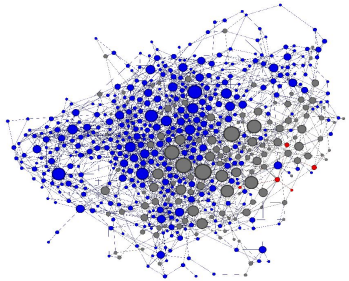
ForceBalance
FF Optimization



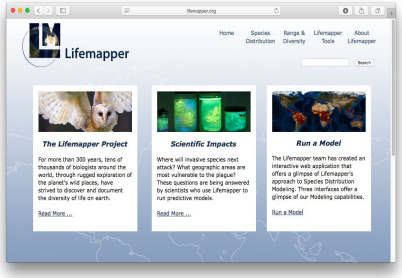
Lobster
CMS Data Analysis

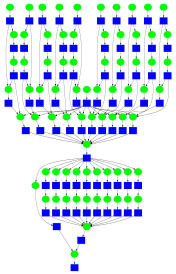


Adaptive Weighted Ensemble
Molecular Dynamics

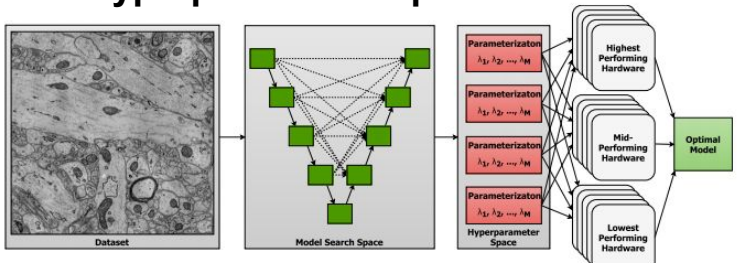


Lifemapper
Species Distribution Modeling



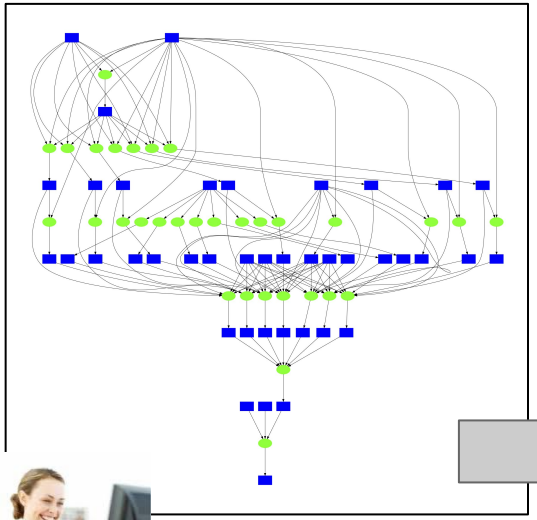


SHADHO
Hyperparameter Optimization

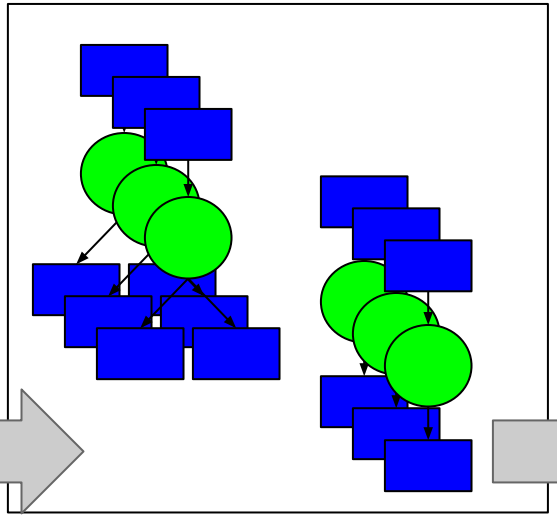


Workflows and Runtime Data Access

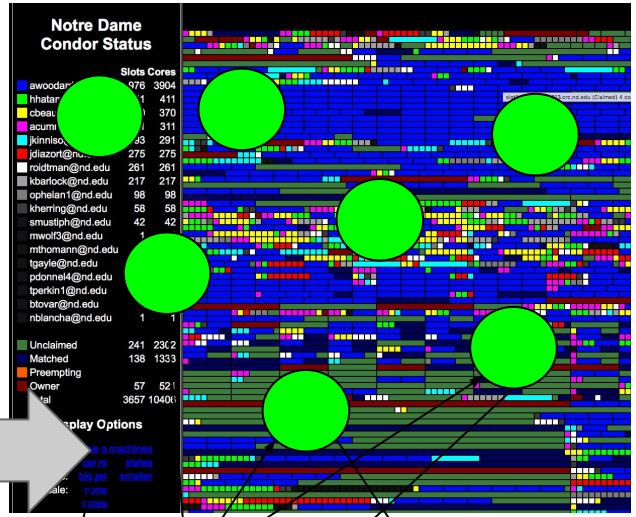
Workflow Manager



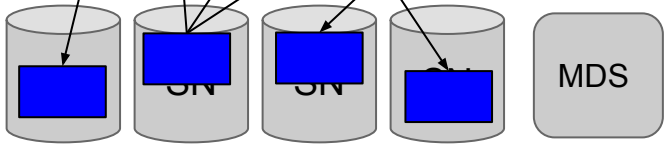
Task / Data Scheduler



Computing Facility



Shared Parallel Filesystem



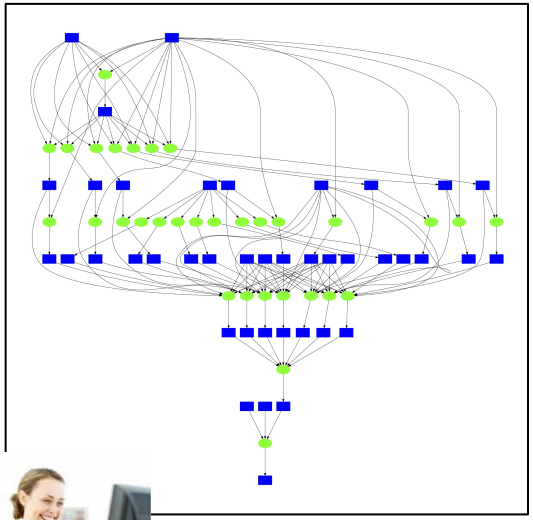
Challenges of Workflows on Clusters

CCTools

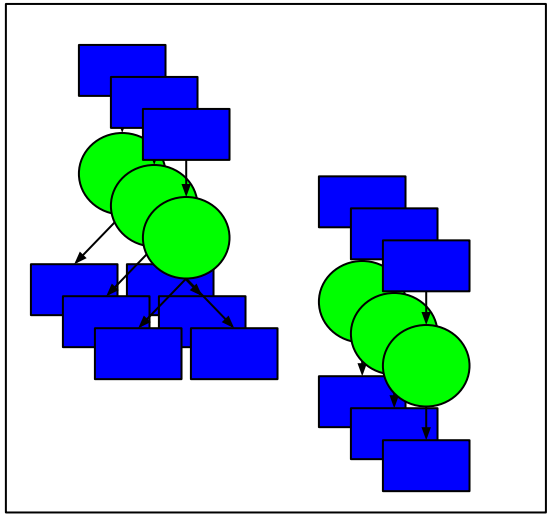
- HPC filesystems are optimized for concurrent large-file access for message-passing jobs: bulk load, coordinated checkpoint, final write.
- **But workflows tend to behave differently:**
  - Traverse deep directory trees of small files. (metadata surge)
  - Access same input file from many nodes at once.
  - Create large intermediate files that are consumed and then deleted.
- **Software is an essential part that is not integrated into the workflow:**
  - conda install tensorflow -> 99 packages, 32K files, 1.2 GB data
  - import tensorflow -> huge startup times at scale due to metadata
  - Same packages get installed and loaded over and over again with small changes, sometimes intended, sometimes not.
- Accelerators have the (positive) effect of decreasing the overall CPU/IO ratio, so it becomes even more essential to place data correctly!

Key Idea: Exploit Storage in Cluster

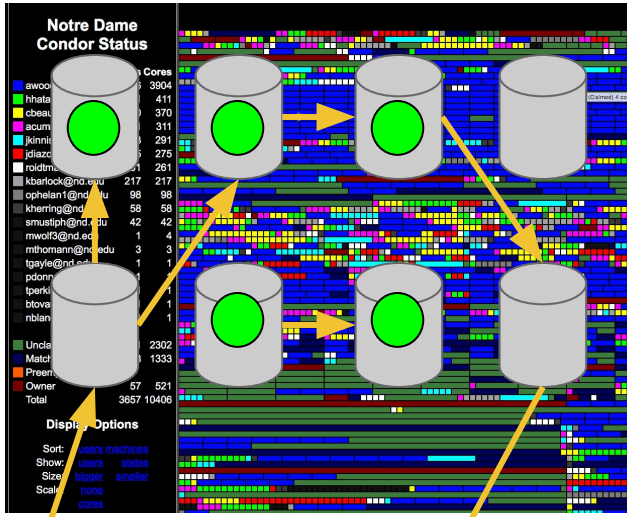
Workflow Manager



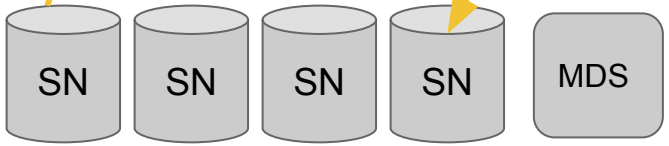
Task / Data Scheduler



Storage Already Embedded in Cluster



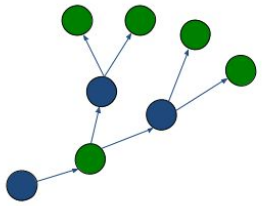
Shared Parallel Filesystem



Outline

- Workflow Systems for Productivity at Scale
- **TaskVine: A Data Intensive Workflow System**
  - **Architecture**
  - Programming Model
  - Data Handling
  - MiniTasks and Serverless
- Applications
- Challenges and Future Work

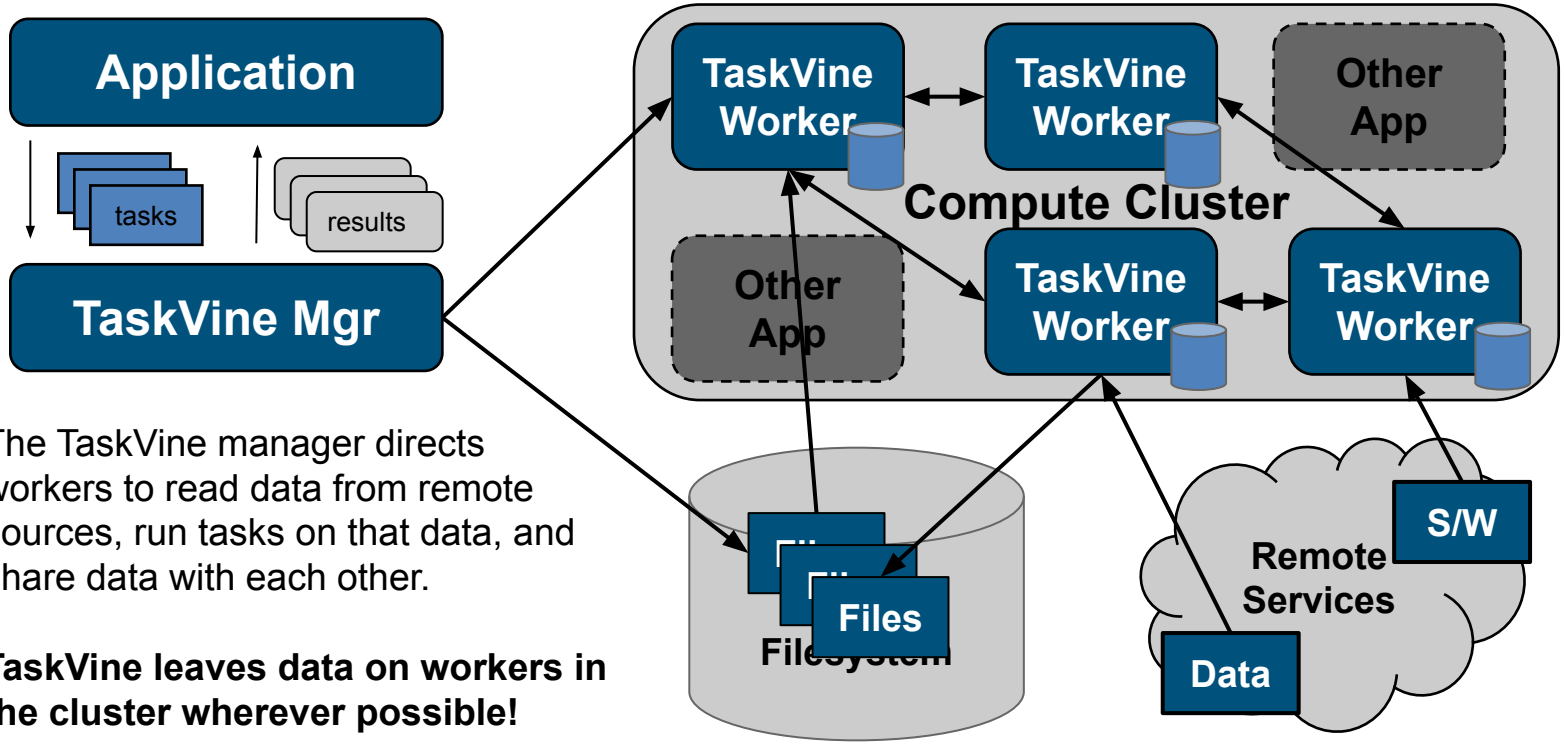
TaskVine



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

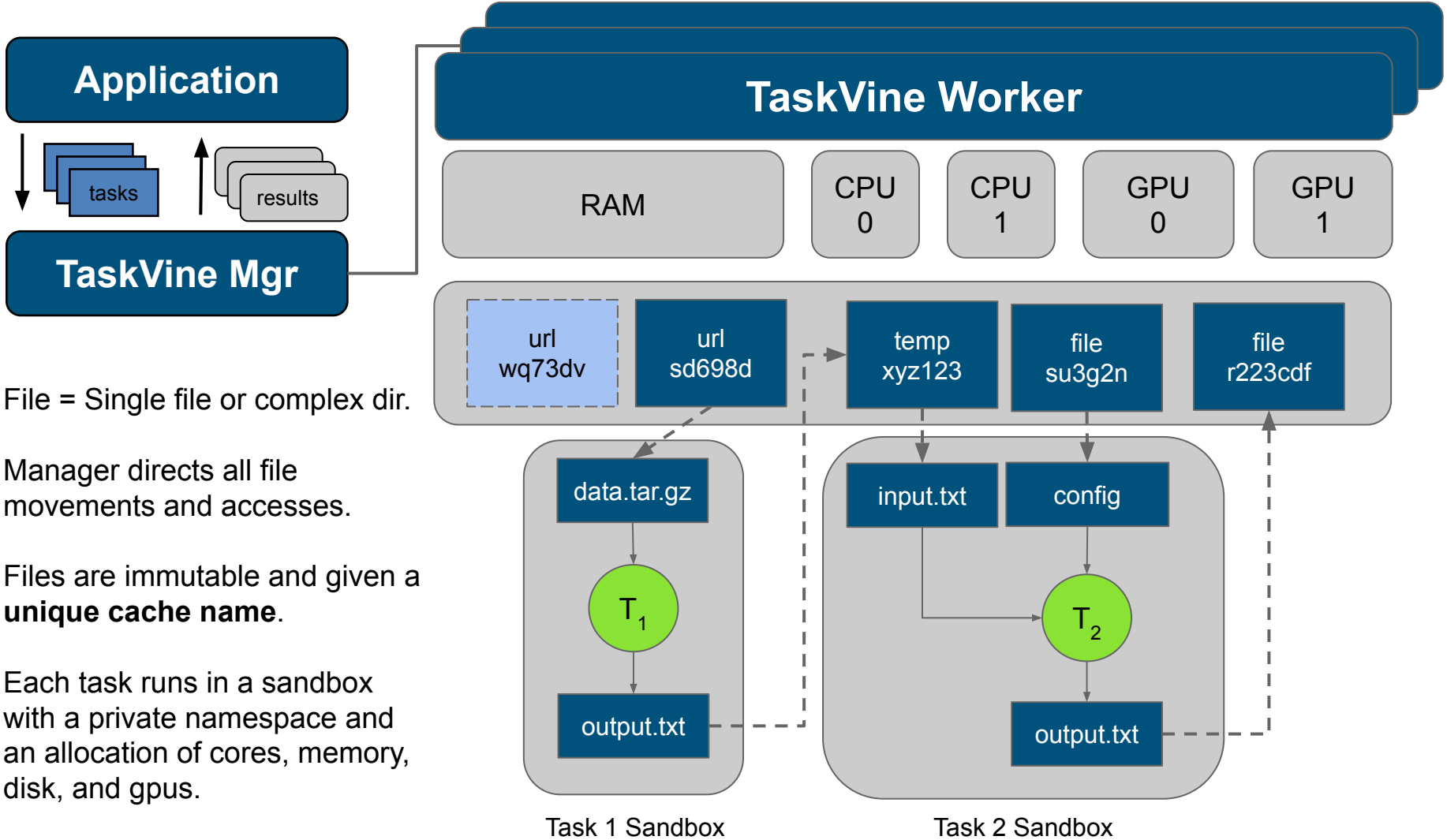


The TaskVine manager directs workers to read data from remote sources, run tasks on that data, and share data with each other.

TaskVine leaves data on workers in the cluster wherever possible!

Design Goals for TaskVine

- **Make it easy** to construct dynamic workflows with thousands to millions of tasks running on thousands of cluster nodes.
- **Handle common failures** by detecting and recovering from worker crashes, network failures, and other unexpected events.
- **Avoid moving data** wherever possible: leave data in place until it needs to be moved or duplicated.
- **Re-use data objects** within and across workflows by tracking provenance from original sources all the way to final outputs.
- **Manage task resources** (cpu, gpu, mem, disk) carefully in order to pack in as much as we can (but not too much!) into each worker.
- **Support complex software** environments built from package managers by explicitly naming dependencies of tasks.



File = Single file or complex dir.

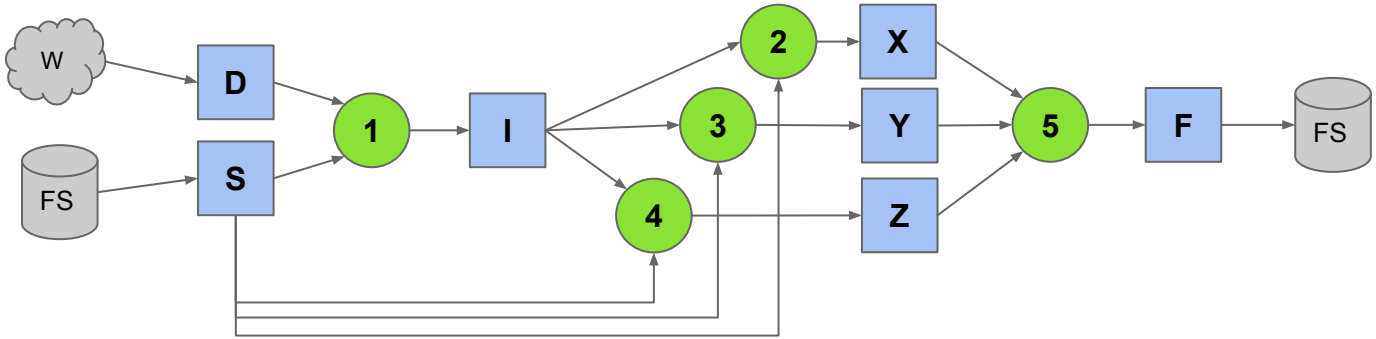
Manager directs all file movements and accesses.

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.

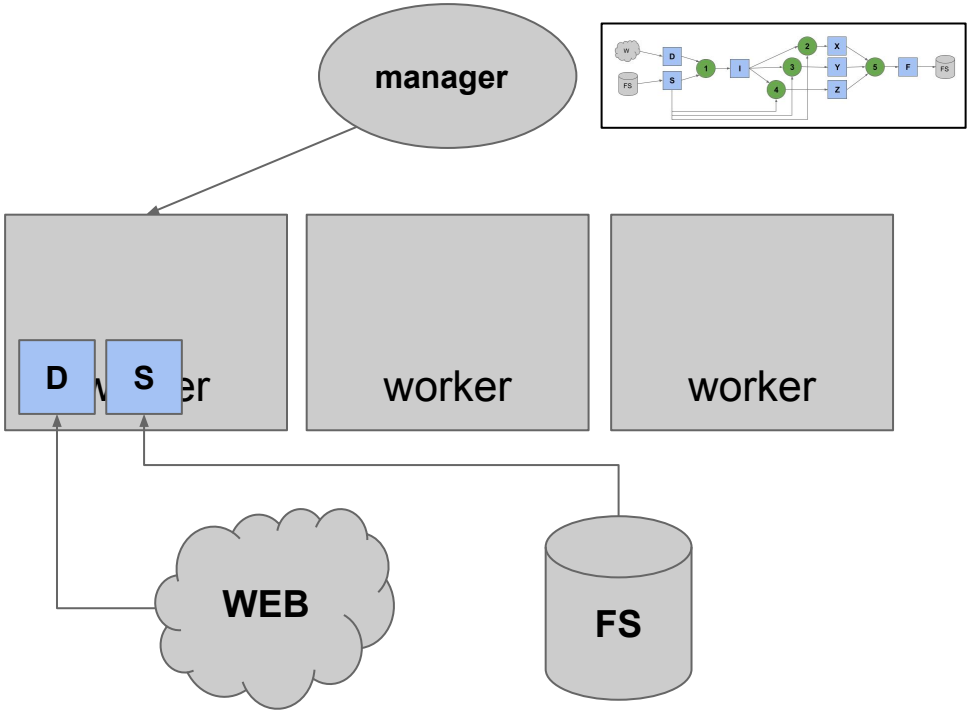
In-Cluster Data Management

Suppose you have a workflow like this: a dataset D comes from a web repository, a software package S comes from the shared filesystem. After passing through tasks 1-5, the final output F should be written to the filesystem. TaskVine aims to keep all of the data within the cluster, as follows.



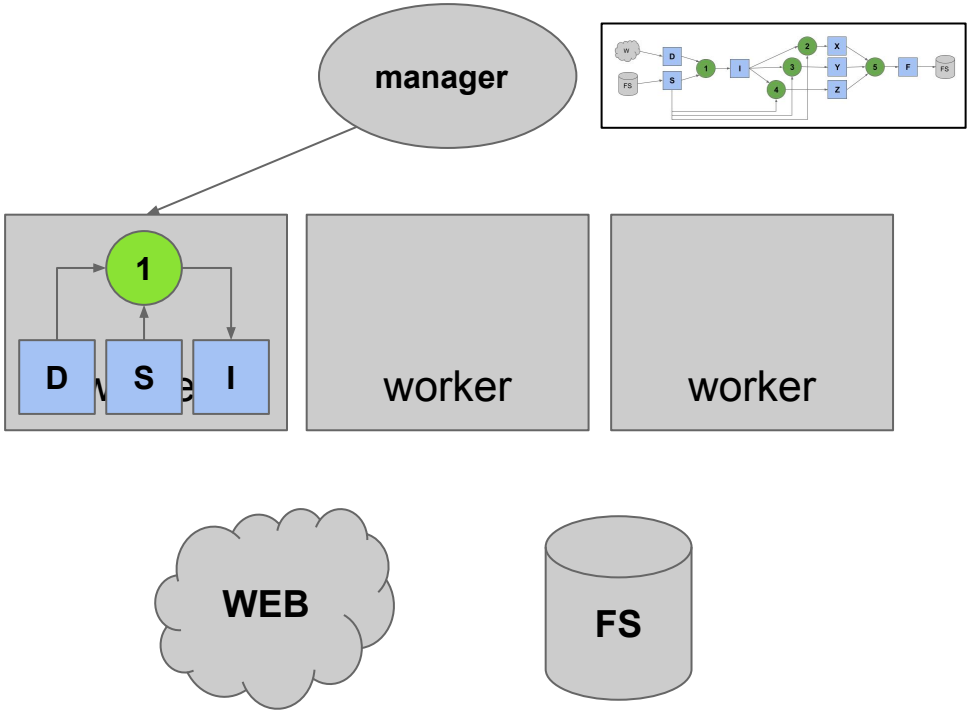
In-Cluster Data Management

The manager selects a worker for task 1, and then directs dataset D to be downloaded from the web, and software package S to be loaded from the shared filesystem.



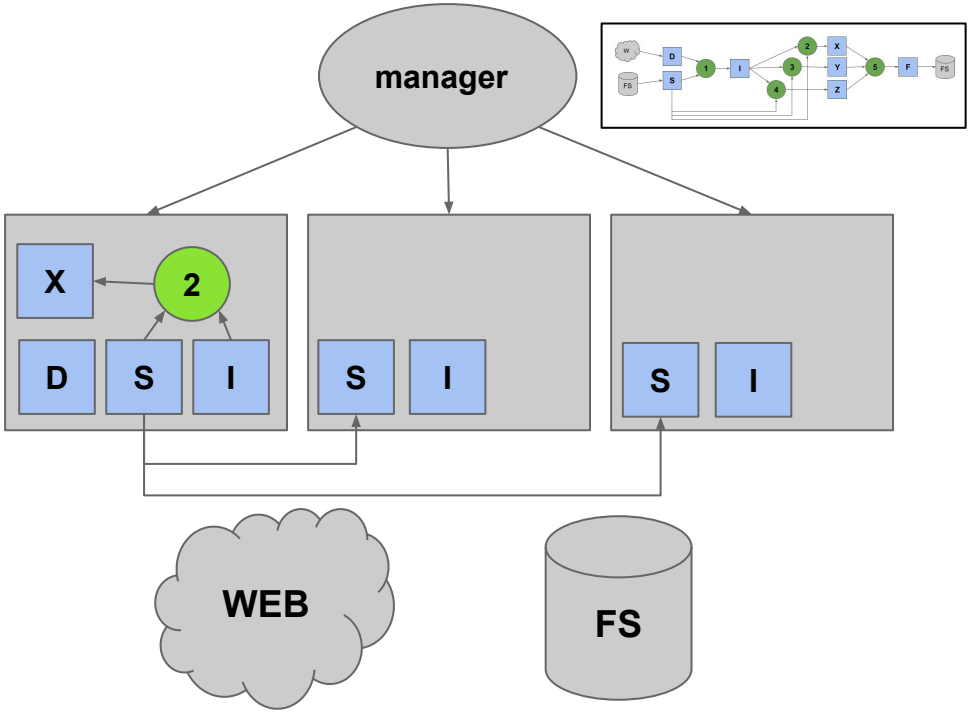
In-Cluster Data Management

Next, task 1 is dispatched to that worker, where it reads dataset D, runs software package S, and produces file I, which stays where it is created.



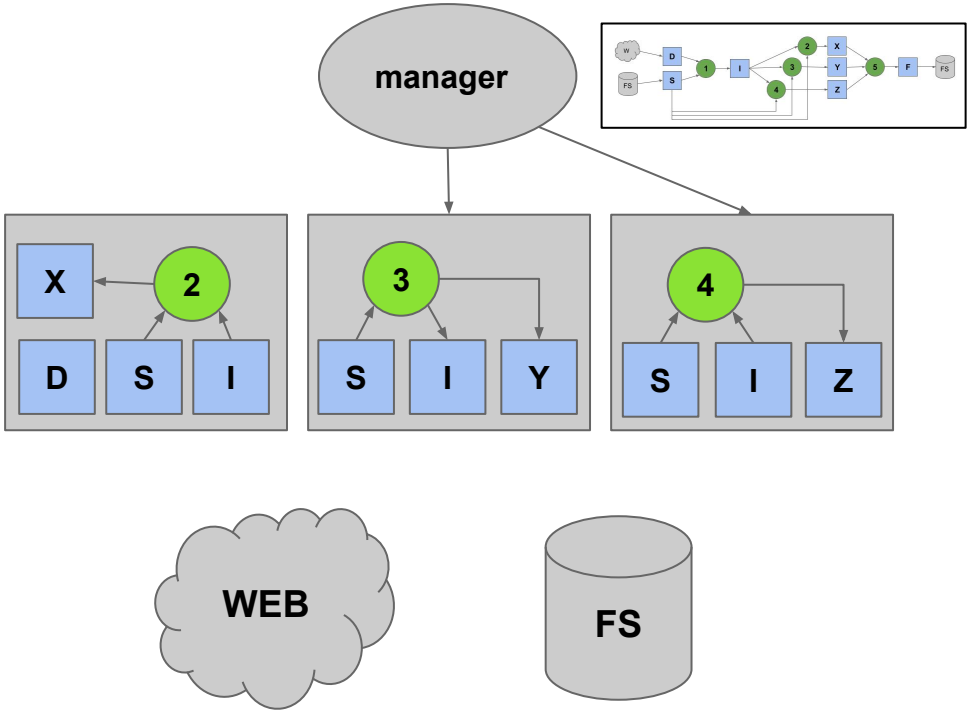
In-Cluster Data Management

Once file I is created, task 2 can run immediately on that node, producing file X. Software package S and file I are duplicated to the other worker nodes.



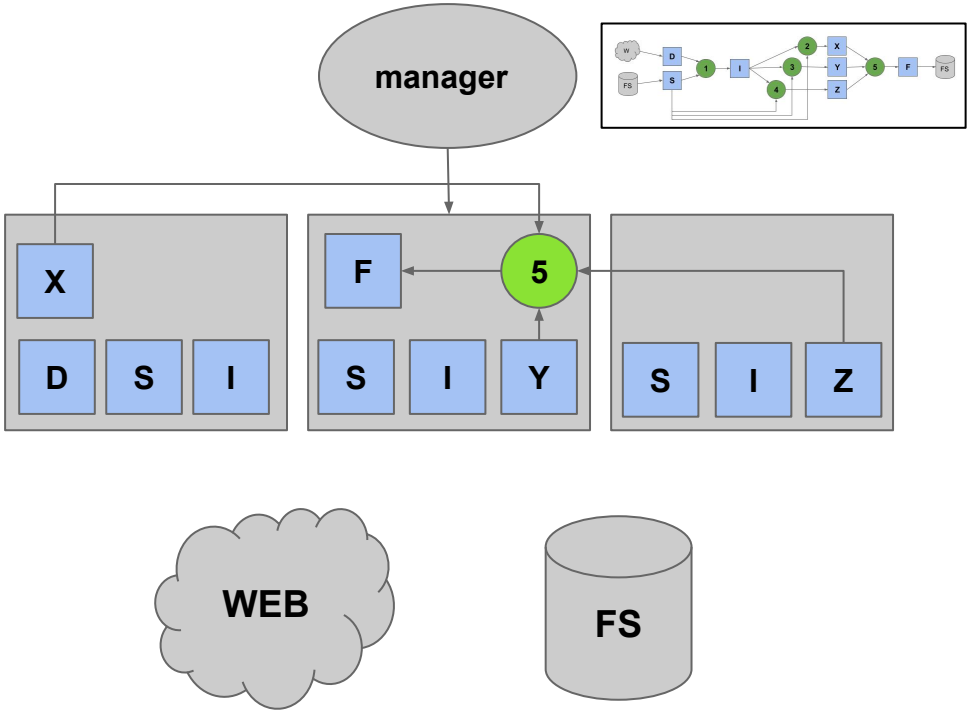
In-Cluster Data Management

Now tasks 3 and 4 can run on the other worker nodes, producing files Y and Z.



In-Cluster Data Management

Next, task 5 is dispatched to the middle worker. It consumes files X, Y, and Z, which are pulled in from peer nodes. The output file X is produced on that node.

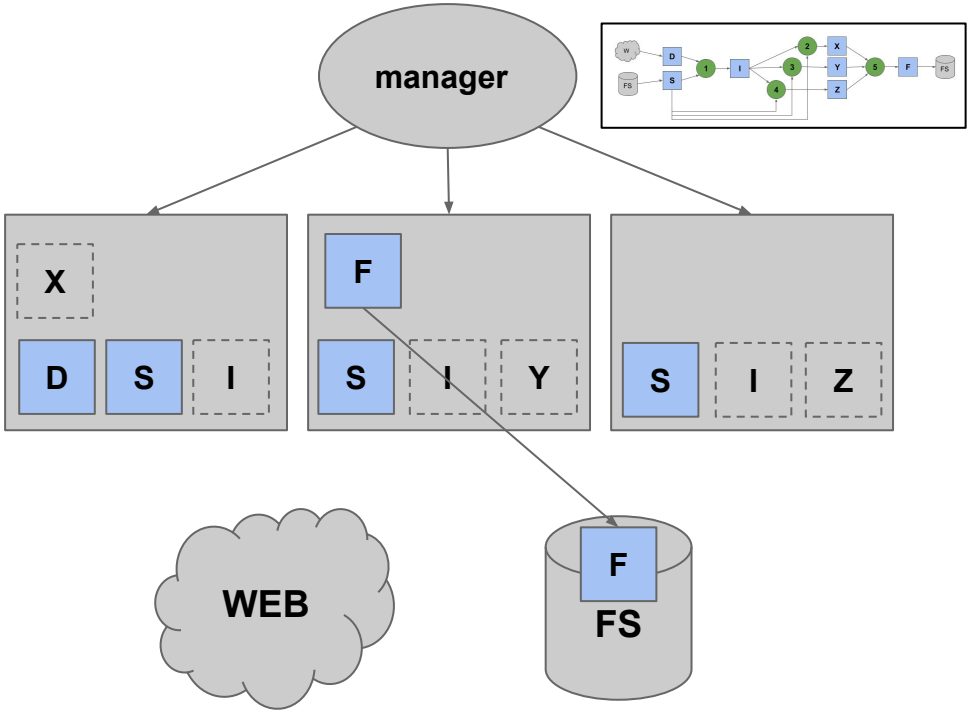


In-Cluster Data Management

Finally, output file F is written back to the shared filesystem, as the ultimate output of the workflow.

The manager directs the workers to delete any remaining uncacheable files.

Common input files remain to accelerate future workflows.



Outline

- Workflow Systems for Productivity at Scale
- TaskVine: A Data Intensive Workflow System
  - Architecture
  - **Programming Model**
  - Data Handling
  - MiniTasks and Serverless
- Applications
- Challenges and Future Work

API: Declare Files Explicitly

```
import 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 )
software = m.declarePoncho( package )
```

API: Connect Tasks to Files

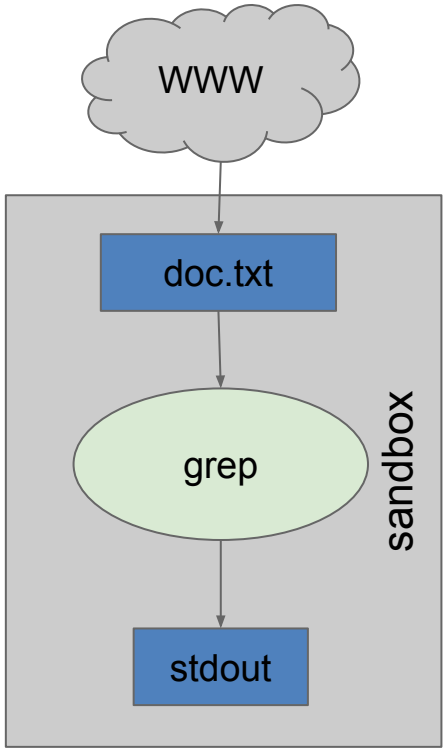
```
task = vine.Task("mysim.exe -p 50 input.data -o output.data")

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: Connect Tasks to Files

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

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)
```

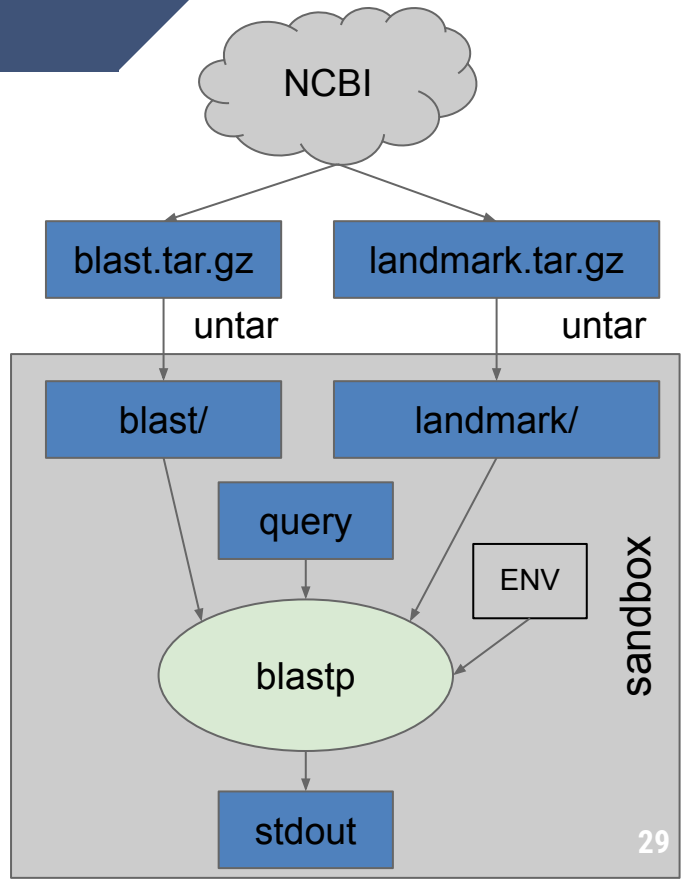
Defining a Simple Task

```
import taskvine as vine

m = vine.Manager(9123)

doc = m.declareURL("https://www.gutenberg.org/files/1960/1960.txt")

task = vine.Task("grep chair doc.txt")
task.add_input(doc,"doc.txt")

taskid = m.submit(task)
task = queue.wait(VINE_FOREVER)

print task.output
```



A Real Application: NCBI Blast

```
blast_url="https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+
/LATEST/ncbi-blast-2.13.0+-x64-linux.tar.gz"

landmark_url =
"https://ftp.ncbi.nlm.nih.gov/blast/db/landmark.tar.gz"

query_string = "GCTAATCCA…"

software = m.declareUntar(m.declareURL(blast_url))
landmark = m.declareUntar(m.declareURL(landmark_url))

task = vine.Task("blastp -db landmark -query query.file")
task.add_input(software,"blastdir")
task.add_input(database,"landmark")
task.add_input_buffer(query_string, "query.file")
task.set_env_var("BLASTDB", value="landmark")

m.submit(task)
```

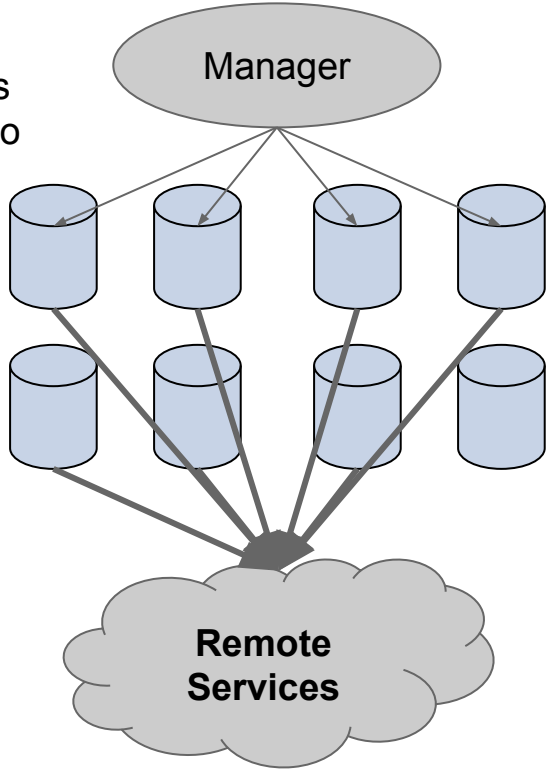


Outline

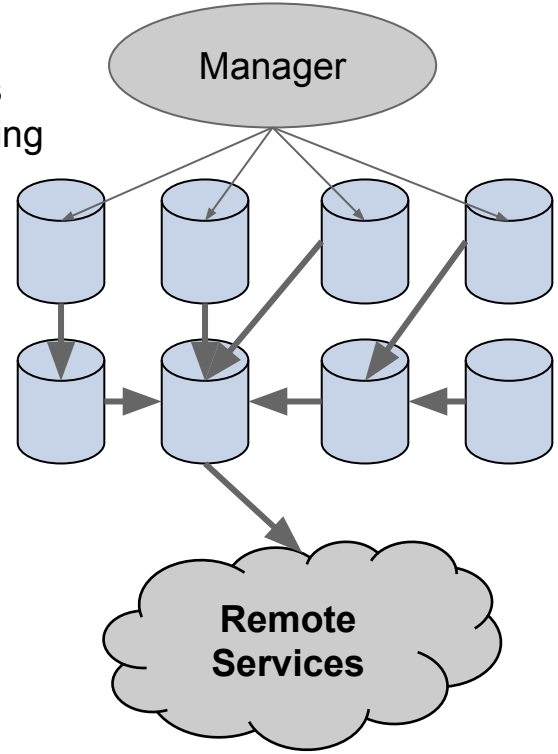
- Workflow Systems for Productivity at Scale
- TaskVine: A Data Intensive Workflow System
  - Architecture
  - Programming Model
  - **Data Handling**
  - MiniTasks and Serverless
- Applications
- Challenges and Future Work

Transfer Management

Uncontrolled:
Manager dispatches unlimited transfers to target data source.



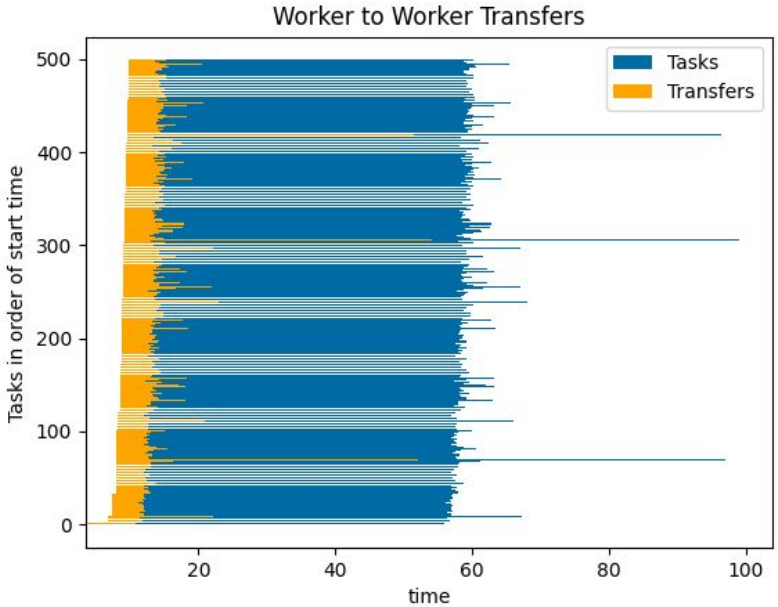
Controlled:
Manager dispatches transfers in a spanning tree with a limited load per node. (default 3)



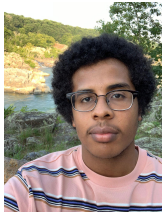
Colin Thomas

Worker to Worker Transfers

BLAST Workflow: Cold start for 500 tasks on 500 workers, each requiring software package and database, ~30s compute each. (Left) Uncoordinated transfers dominate execution time. (Right) Coordinated transfers between workers distribute data exponentially.



Generating Unique Cacheable Names

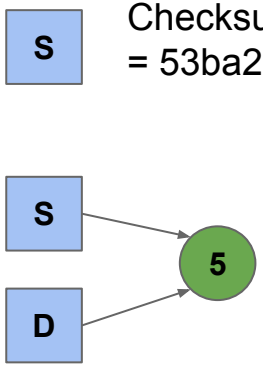


Barry Sly-Delgado

Files have one of three lifetimes:

- single-task
- workflow (default)
- forever

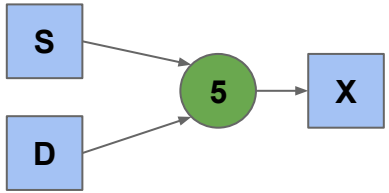
"forever" cached objects are given
content addressable names from a **Merkle Tree** of the file's provenance. If any inputs change, then so does the name of the output, and it's not the same file.



Checksum(Content-of-S)
= 53ba27f

Checksum(
```
"cmd":"blast",
"inputs" = {
  "S" : Checksum(S),
  "D": Checksum(D)
}
```
)

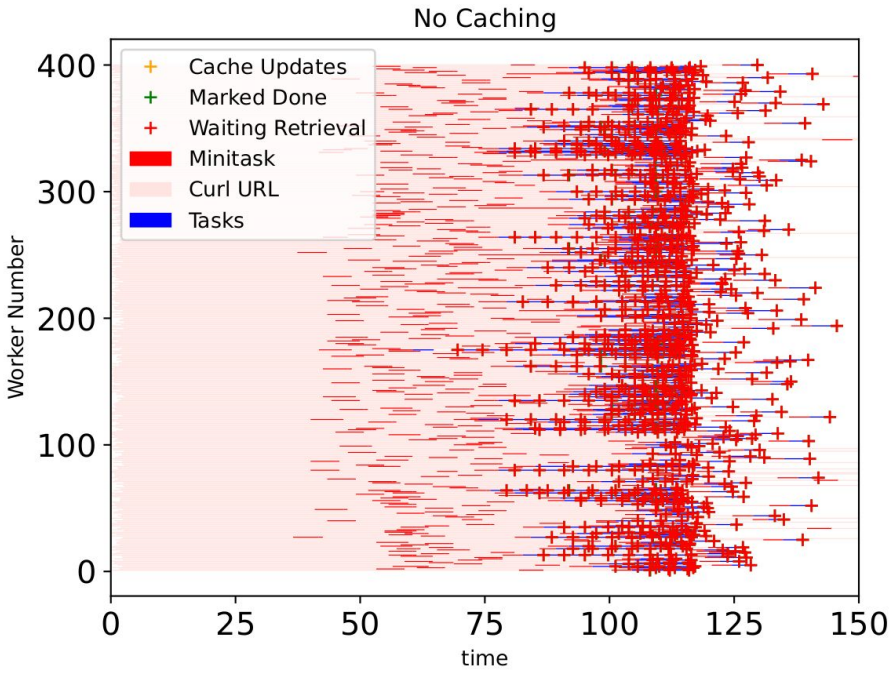
= c320b61

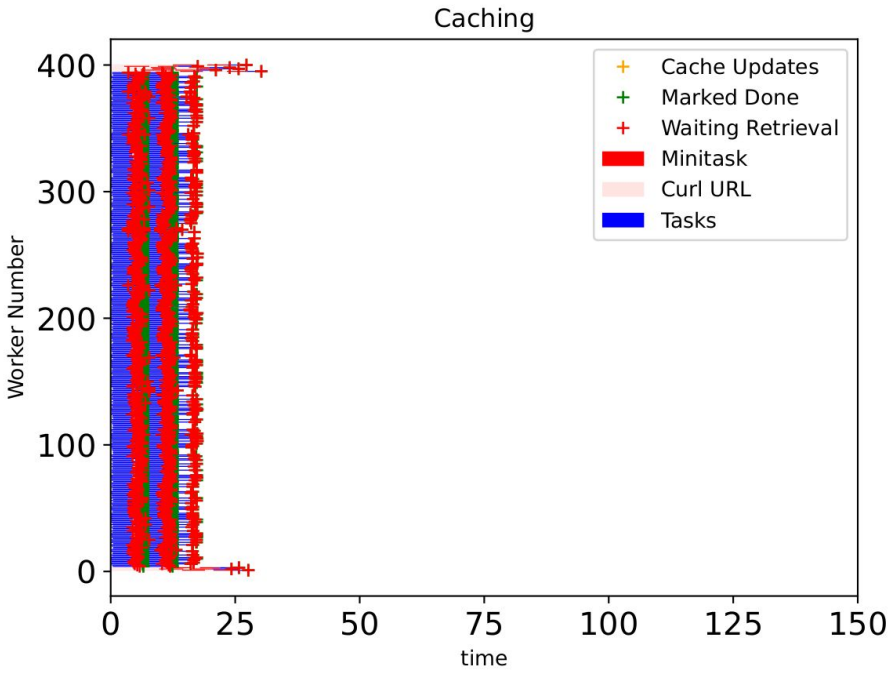


Checksum(
```
"task" : Checksum(Task-5)
"output": "X"
```
)

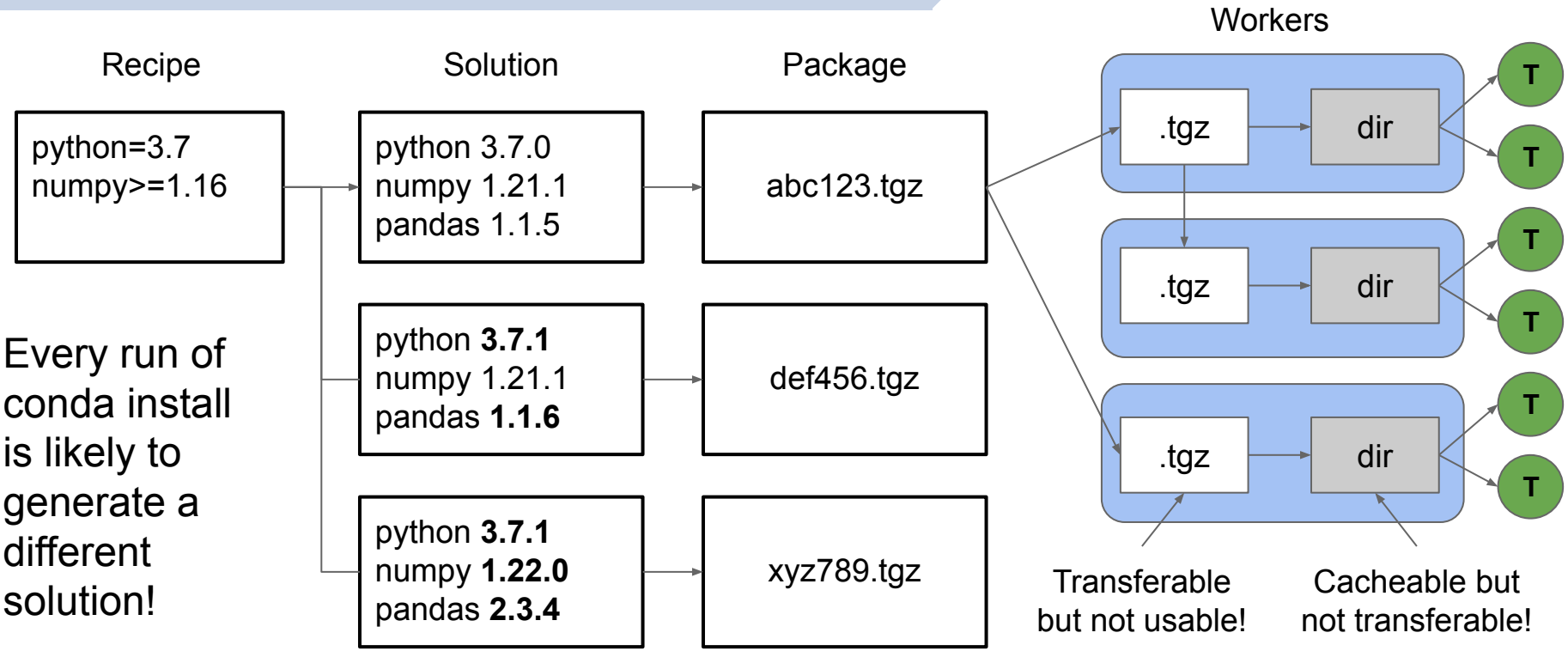
= f06da39

Eliminating Startup Costs





Sharing Software Environments



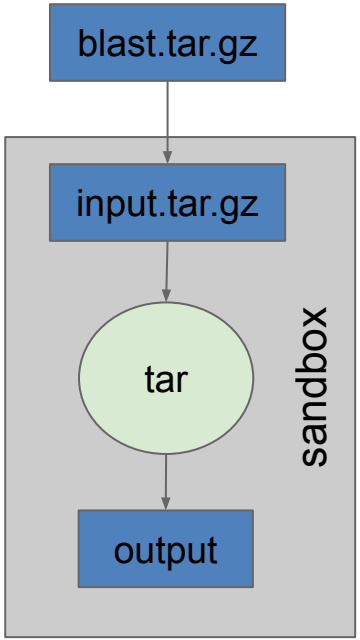
Every run of conda install is likely to generate a different solution!

Outline

- Workflow Systems for Productivity at Scale
- TaskVine: A Data Intensive Workflow System
  - Architecture
  - Programming Model
  - Data Handling
  - **MiniTasks and Serverless**
- Applications
- Challenges and Future Work

Mini-Tasks: FileUntar(f)

```
blast_url="https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+
/LATEST/ncbi-blast-2.13.0+-x64-linux.tar.gz"

landmark_url =
"https://ftp.ncbi.nlm.nih.gov/blast/db/landmark.tar.gz"

query_string = "GCTAATCCA…"

software = m.declareUntar(m.declareURL(blast_url))
landmark = m.declareUntar(m.declareURL(landmark_url))

task = vine.Task("blastp -db landmark -query query.file")
task.add_input(software,"blastdir")
task.add_input(database,"landmark")
task.add_input_buffer(query_string, "query.file")
task.set_env_var("BLASTDB", value="landmark")
```



FileUntar

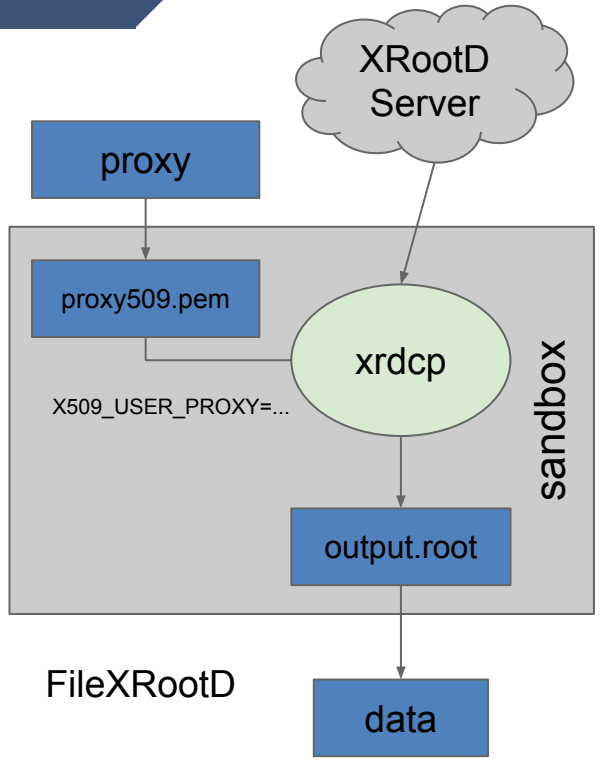
Mini-Task: FileXRootD

New capabilities are added to the system by defining mini-tasks that use the same task infrastructure to define dependencies and execute them reproducibly:

```
data = m.declareXRootD( "xrootd://host/path", "proxy" )
```

Which is defined as a mini-task like this:

```
t = vine.Task("xrdcp {} output.root".format(url));
t.add_input(proxy,"proxy509.pem")
t.set_env_var("X509_USER_PROXY","proxy509.pem")
data = m.declareMiniTask(t,"output.root")
```



Mixed Modality Workflows

CCTools

Standard Task:

Define once, runs on any available worker.

Any Unix process with command line args.

Produces output files.

Library Task:

Define once, runs on **all** available workers.

Any Unix process with a JSON invok. protocol.

Implements Func Call.

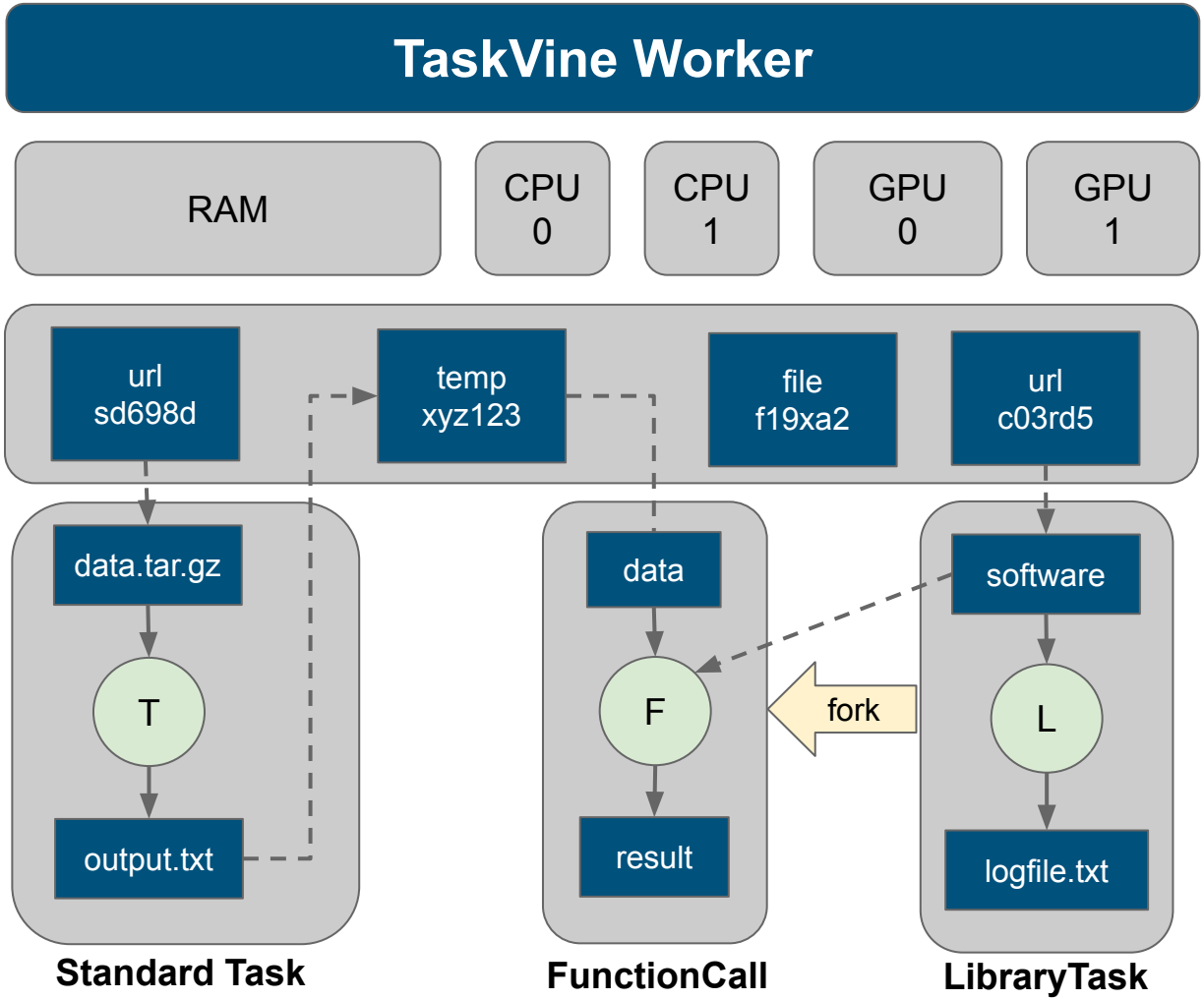
Function Call:

Define once, runs on any matching Library.

JSON definition of args, funcname, and results.

Produces JSON result.

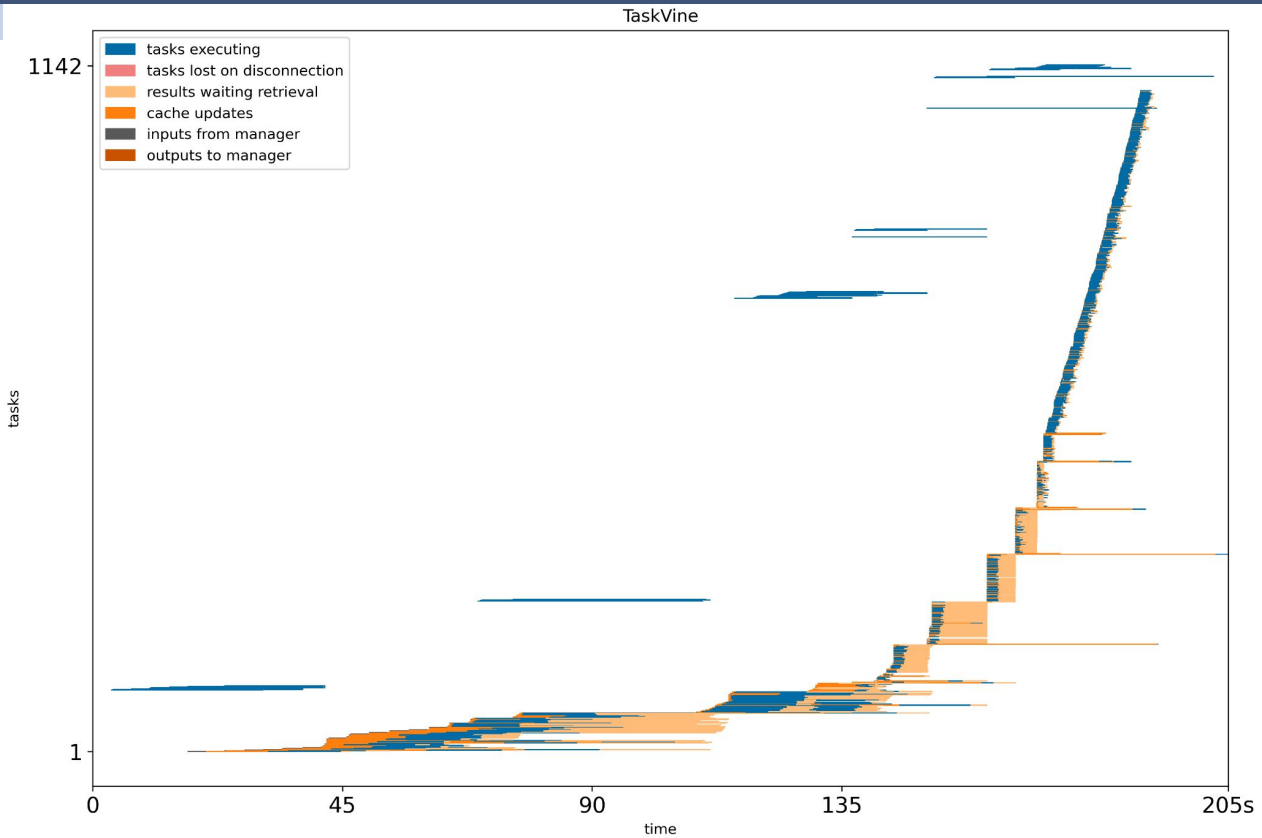
Common Task Capabilities: Resource allocation and management, fault-tolerance, distributed data handling, scheduling, logging, visualization…



Simply converting "import tensorflow" into the preamble of a Library task saves **1.2GB** of Python libraries, **30K** metadata system calls, and **5-10s** latency per FunctionCall. We can mix standard Tasks, Libraries, and FunctionCalls in the same workflow:

David Simonetti

Multi-Model Workflows



100x Standard Tasks
Build model from MNIST data.

For each produced model:
Deploy LibraryTask for inference.

Submit 10x FunctionCalls that invoke each LibraryTask.

Application gradually accelerates as standard tasks produce data that define libraries that can then be invoked.

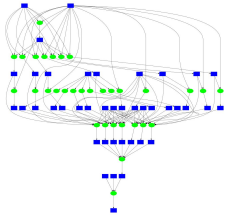
Outline

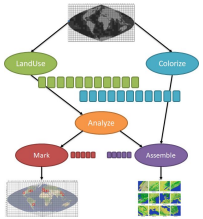
- Workflow Systems for Productivity at Scale
- TaskVine: A Data Intensive Workflow System
  - Architecture
  - Programming Model
  - Data Handling
  - MiniTasks and Serverless
- **Applications**
- Challenges and Future Work

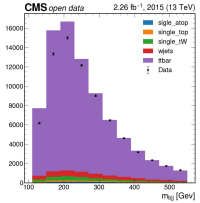
TaskVine is a Workflow Executor

TaskVine can be used directly by custom-written applications that desire fine grained control or it can serve as an execution platform for higher-level workflow languages and systems.

```
import taskvine

file = URL(www)
m.submit(task)
task = m.wait(5)
```







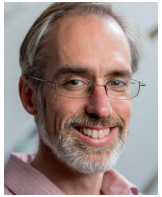
Custom App	Makeflow	Parsl	Coffea
C or Python	C	Python	Python

TaskVine Manager

TaskVine Worker | TaskVine Worker | TaskVine Worker | TaskVine Worker | TaskVine Worker

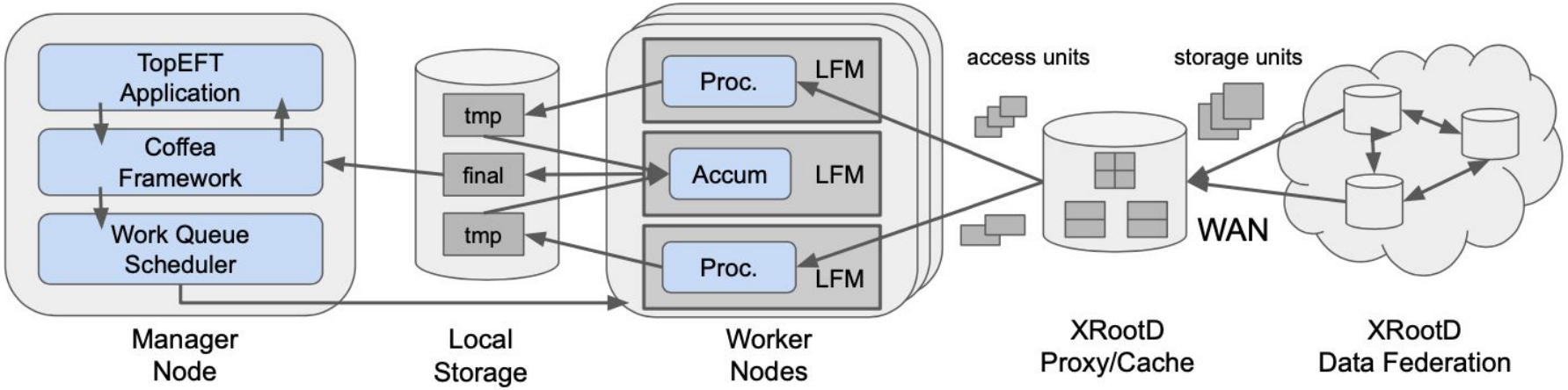
HPC Cluster

Application: TopEFT in WQ

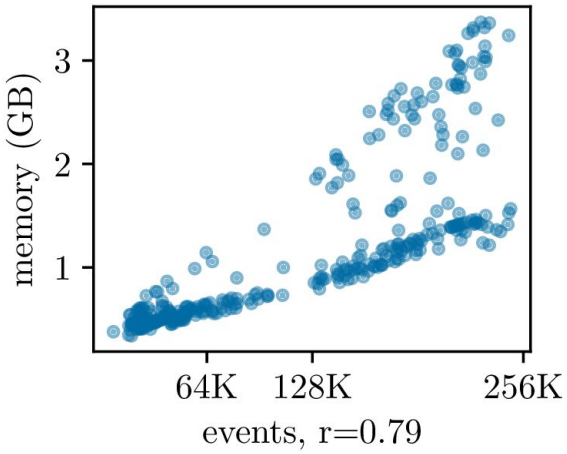


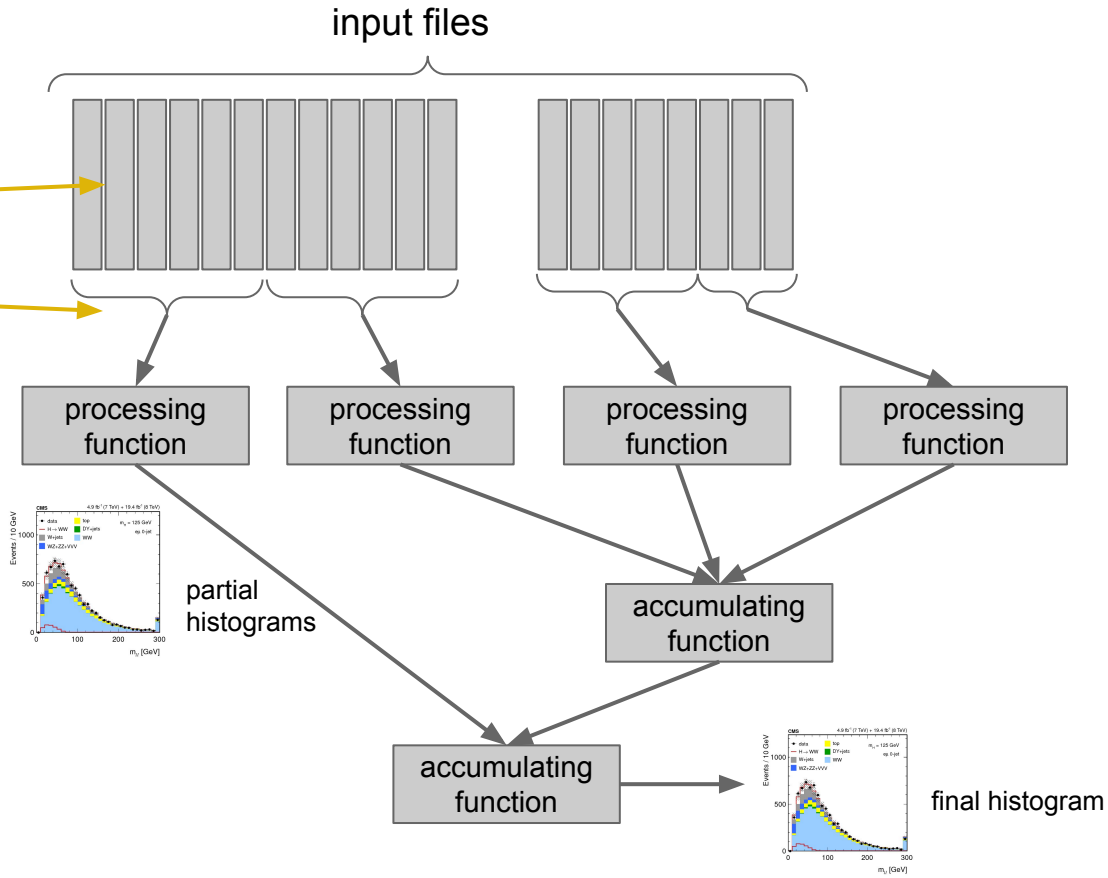
Kelci Mohrman **Kevin Lannon**

- Late stage data analysis for LHC CMS experiment. Search for new physics impacting associated top quark production using the framework of effective field theory (EFT). Custom processing and accumulation functions expressed in Coffea framework, then dispatched using Work Queue. (Our previous system) Remote data access via worldwide XRootD federation, temporaries moved home.



TopEFT / Coffea Data Splitting Workflow

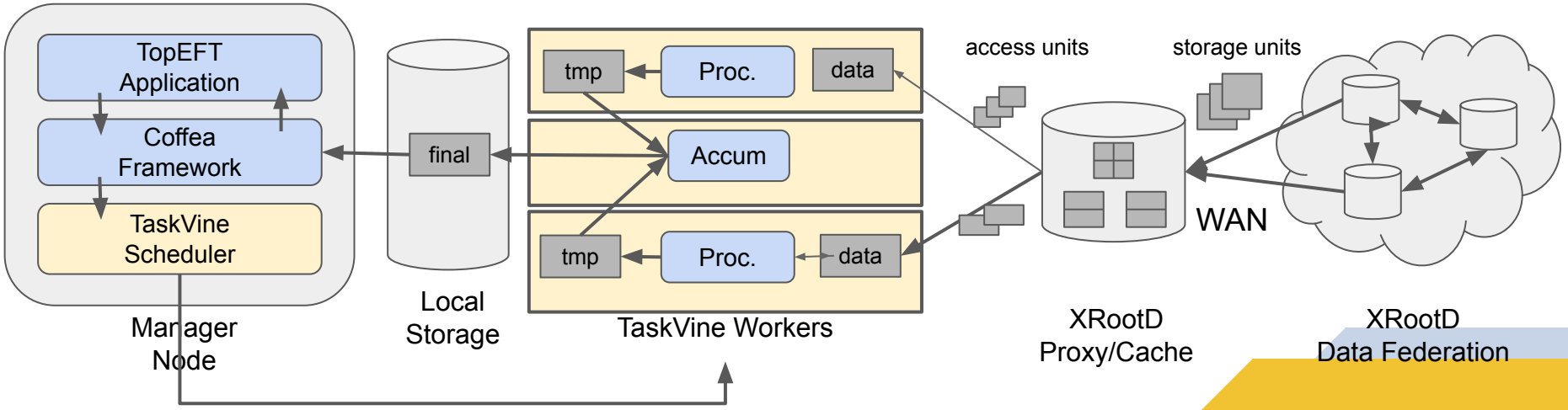




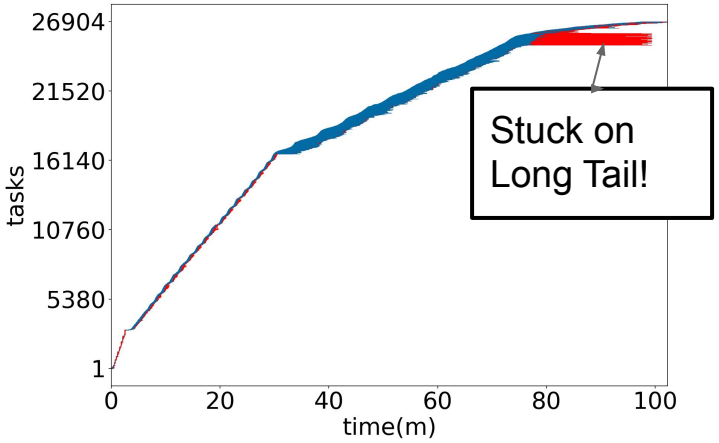
WIP: TopEFT in TaskVine

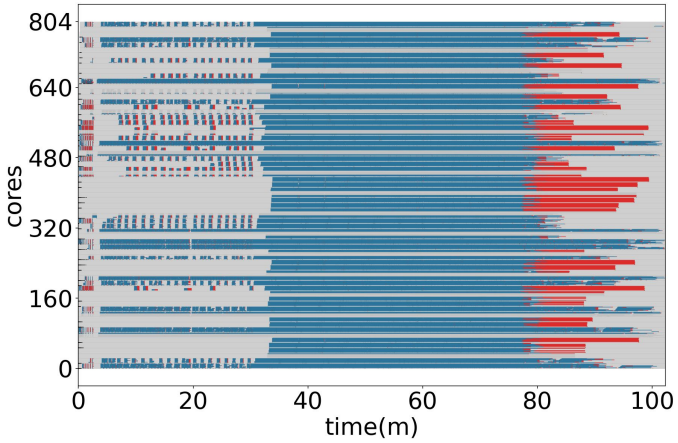
Andrew Hennessee

- New executor module defined to use TaskVine: software and data dependencies are now explicitly declared, and temporaries maintained within the cluster without moving them back.

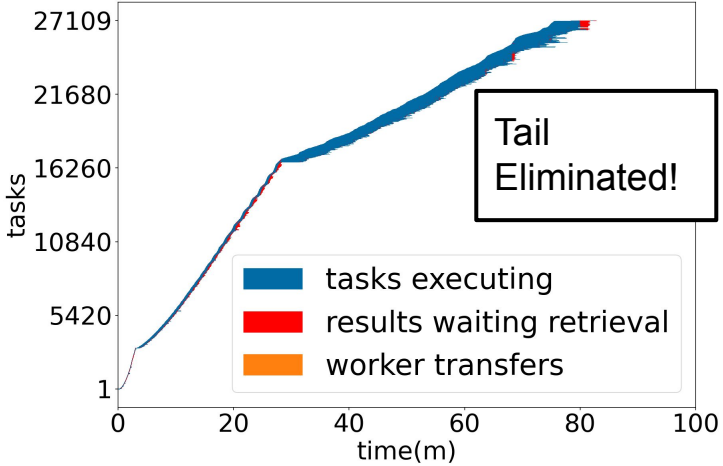


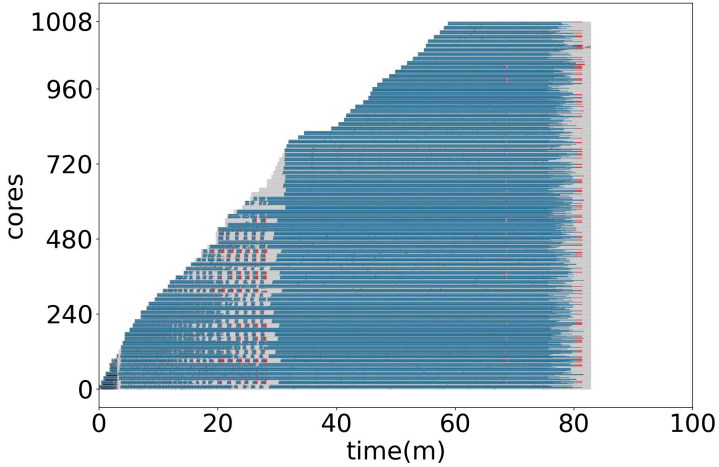
Old:
Accumulation
Data Returned
TopEFT
+ Work Queue





New:
In-Cluster
Accumulation
TopEFT
+ TaskVine





WIP: Parsl + TaskVine

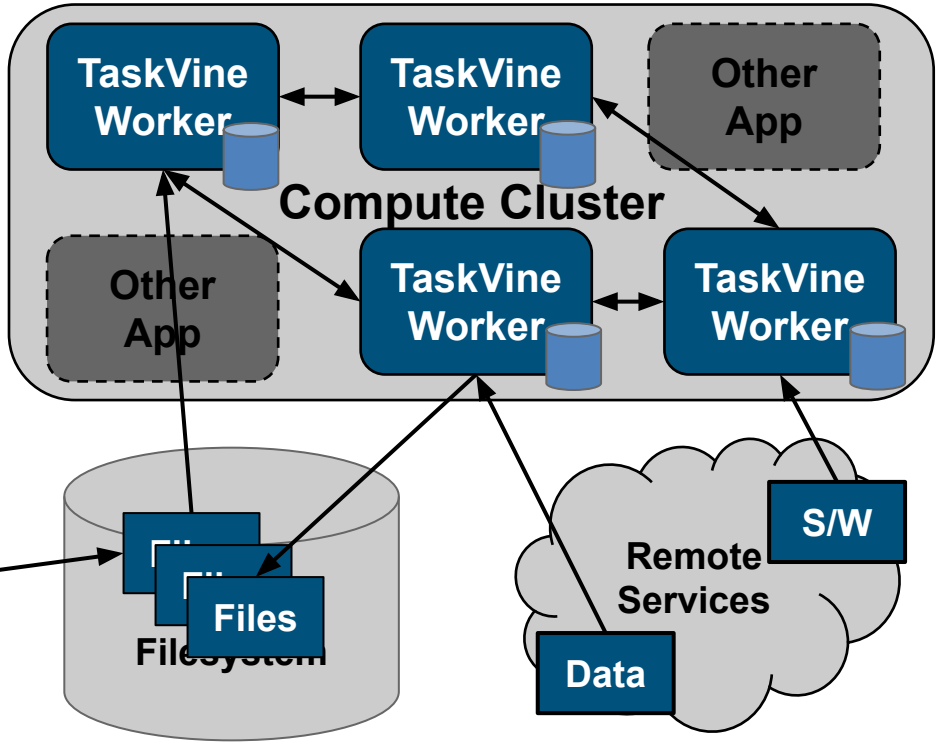
Kyle Chard
U. Chicago

https://parsl-project.org

Parsl Data Flow Kernel

Parsl + TaskVine Exec

TaskVine Mgr



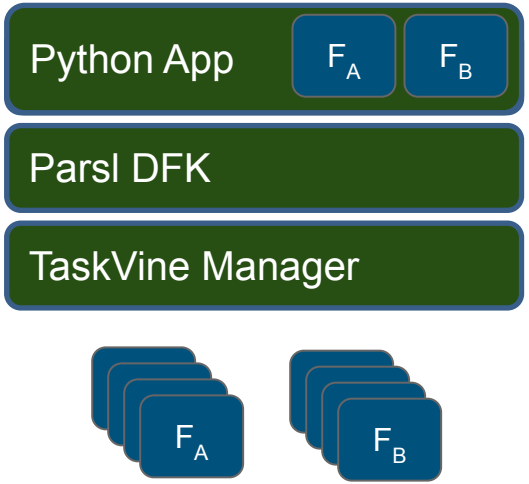
Common Challenges

Two common problems of scaling up:

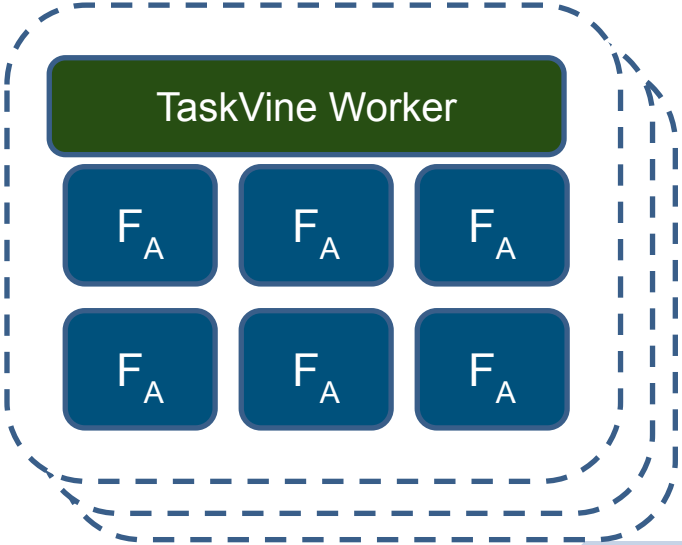
- What resources should be assigned to a function call?
- What software dependencies does this function need?

How can we solve these problems automatically at runtime, without requiring the user to make advance declarations?

Packing Functions Into Manycore Nodes

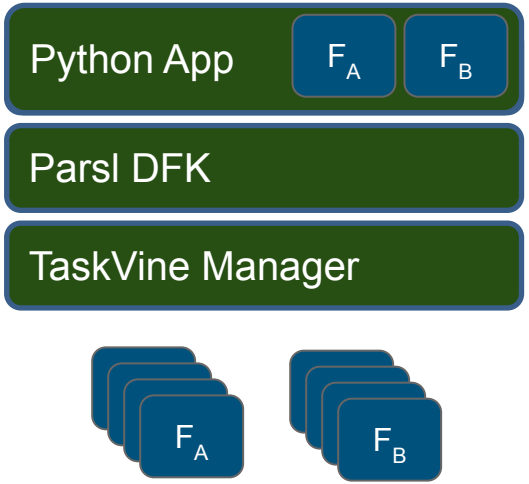


Allocate 2GB per Function A?

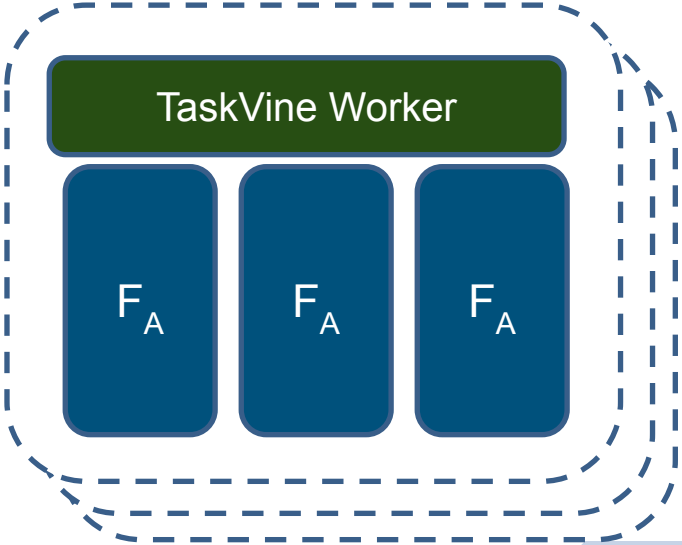


12 cores and 12 GB RAM

Packing Functions Into Manycore Nodes



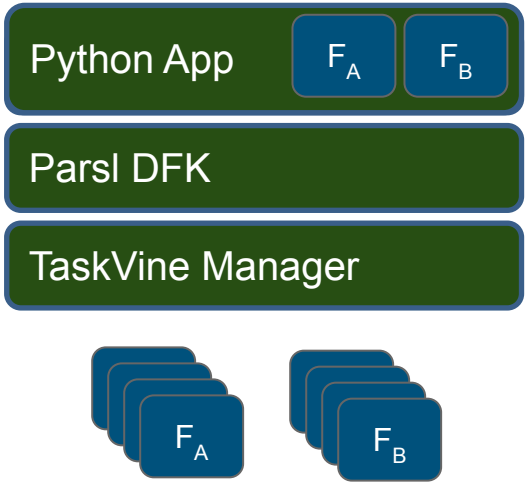
Allocate 4GB per Function A?

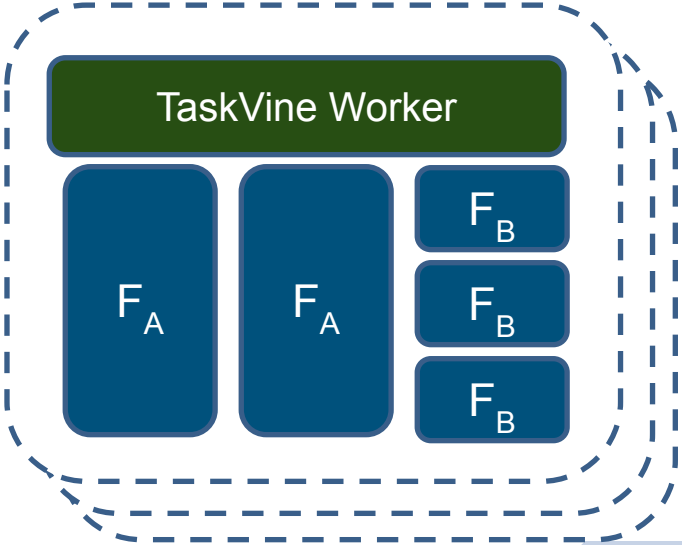


12 cores and 12 GB RAM

Packing Functions Into Manycore Nodes

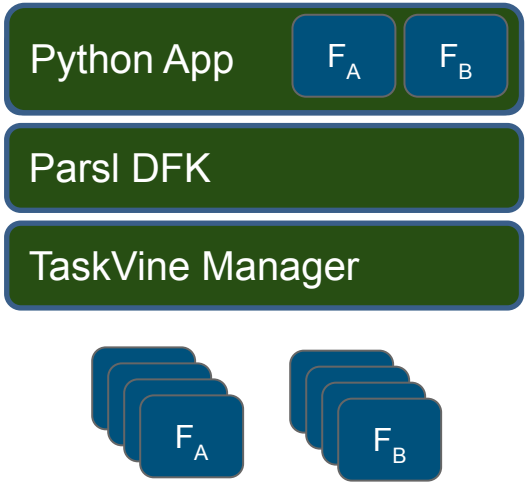
Mix Function A and Function B?



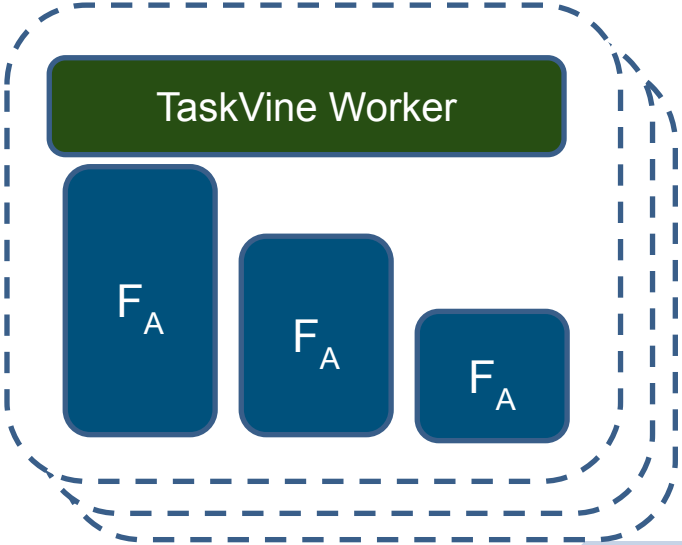


12 cores and 12 GB RAM

Packing Functions Into Manycore Nodes



What if Function A Varies?

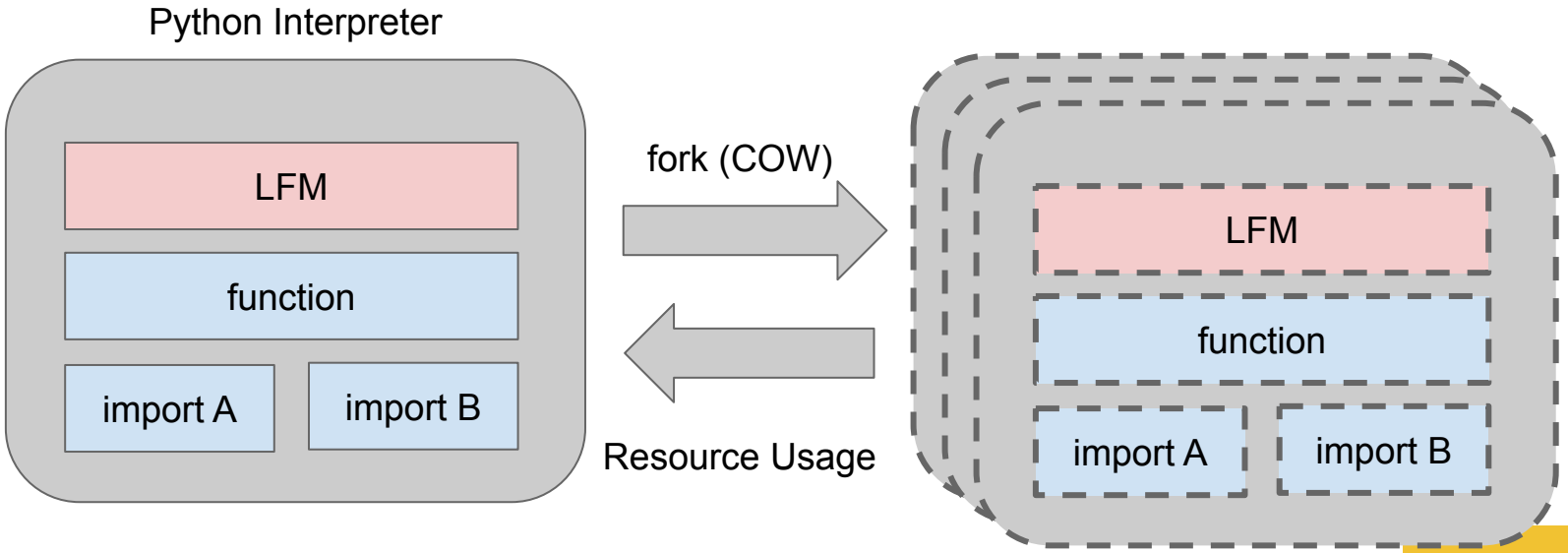


12 cores and 12 GB RAM

How to measure a single function call?

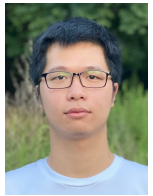
Ben Tovar

LFM - Lightweight Function Monitor

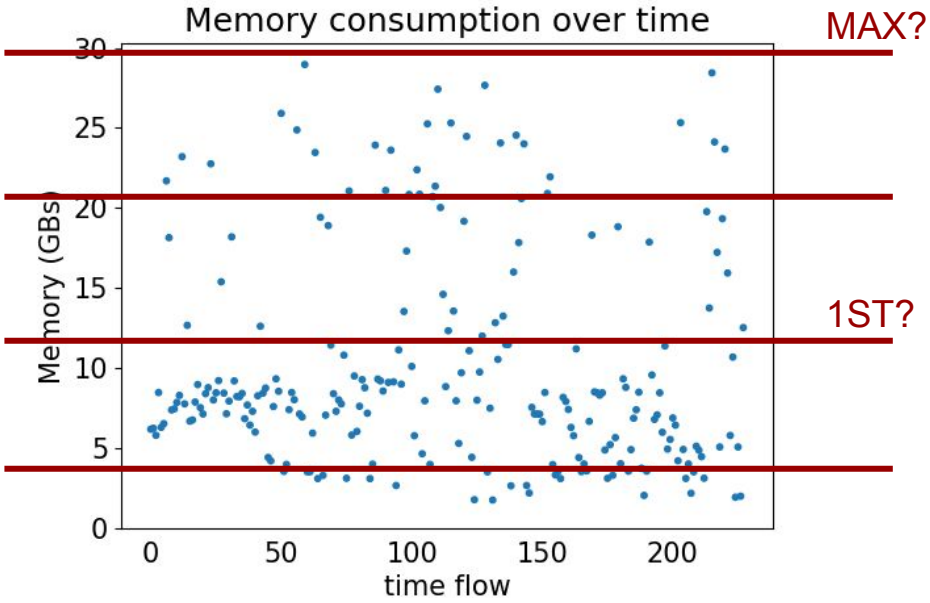


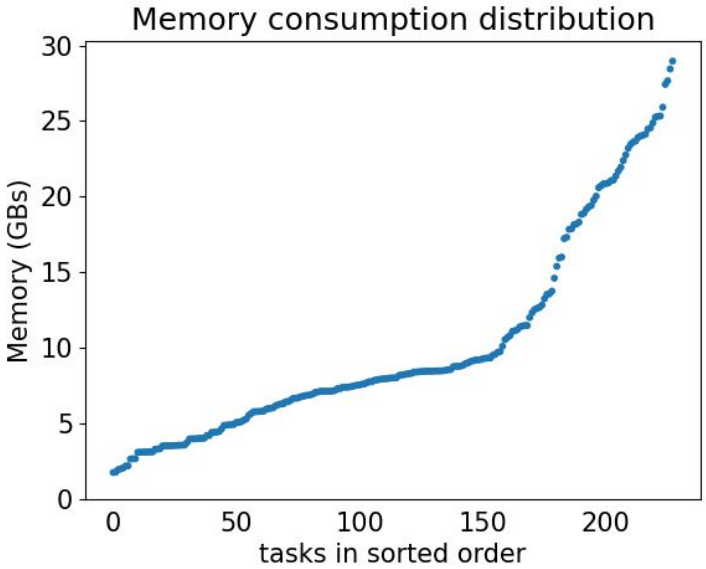
Tim Shaffer, Zhuozhao Li, Ben Tovar, Yadu Babuji, TJ Dasso, Zoe Surma, Kyle Chard, Ian Foster, and Douglas Thain, **Lightweight Function Monitors for Fine-Grained Management in Large Scale Python Applications**, IEEE International Parallel & Distributed Processing Symposium, May, 2021. DOI: 10.1109/IPDPS49936.2021.00088

Example: Memory Usage in Colmena-XTB



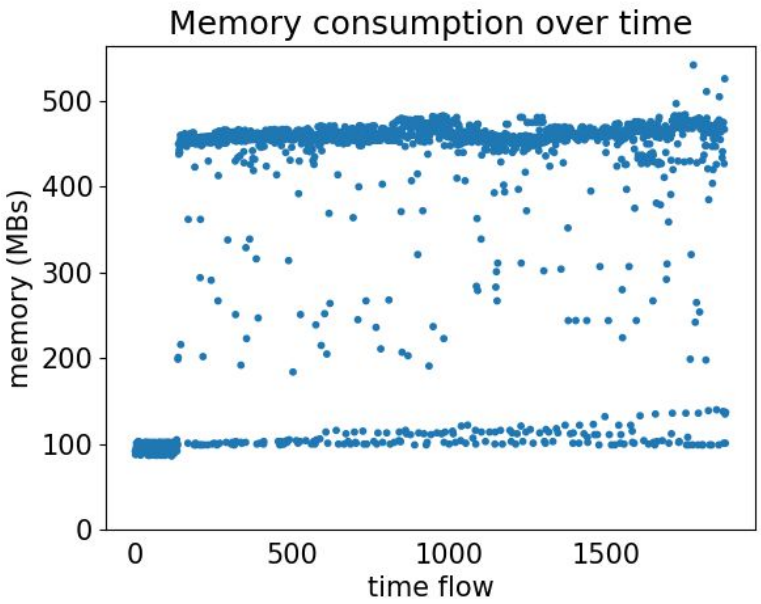
Thanh Phung

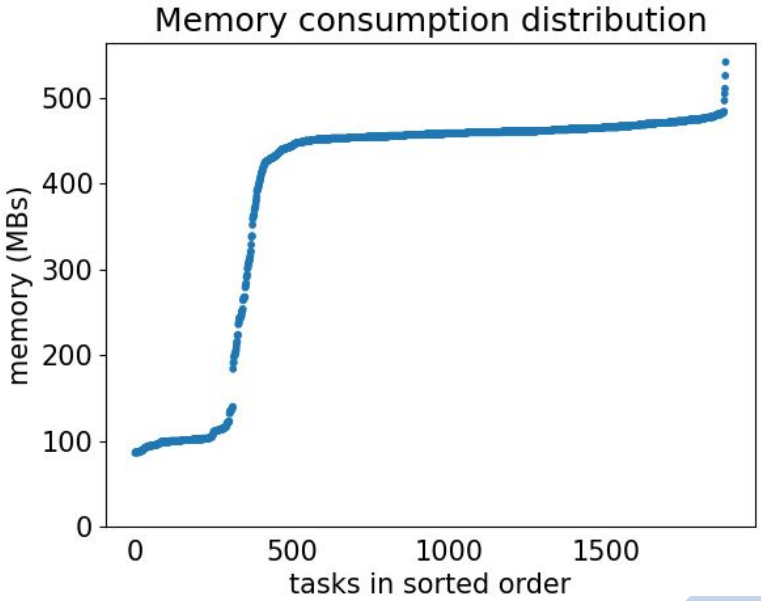




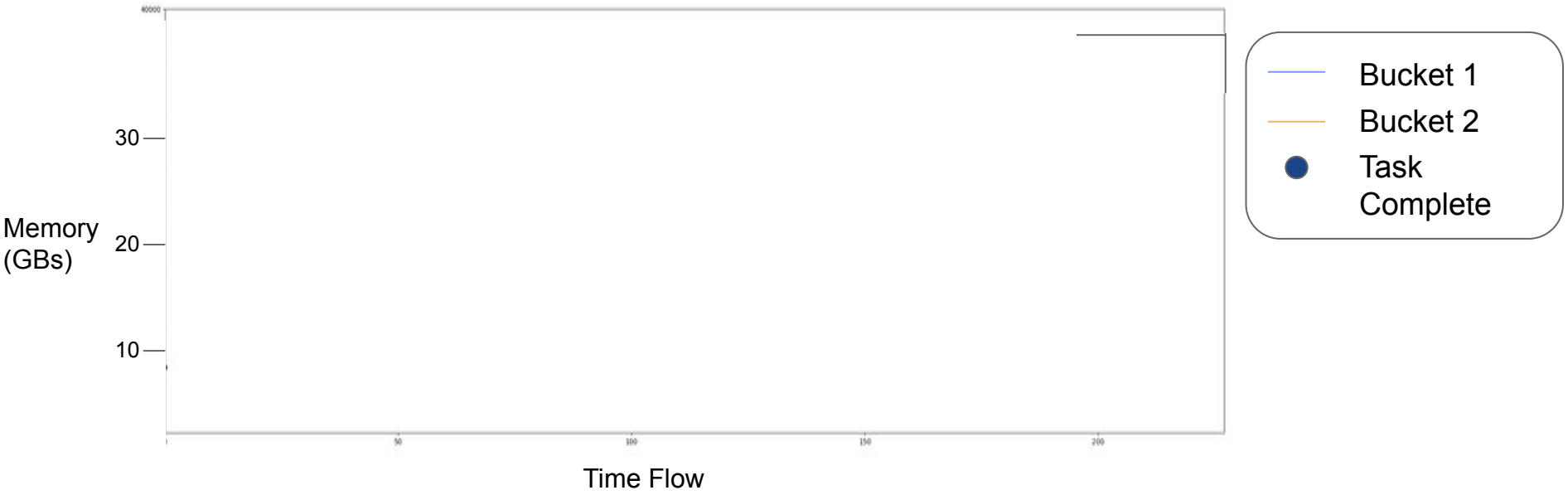
Thanh Son Phung, Logan Ward, Kyle Chard, Douglas Thain, "**Not All Tasks are Created Equal: Adaptive Resource Allocation for Heterogeneous Tasks in Dynamic Workflows**", WORKS Workshop at Supercomputing 2021.

Example: Memory Consumption in TopEFT





Dynamic K-means Bucketing



Evaluation - Average Task Efficiency

	Colmena	TopEFT	Normal	Uniform	Expone-ntial	Bimodal	Trimodal
Whole Machine	15.8	0.6	12.4	39.1	15.7	31.3	30.7
Double Allocation	**51.9**	4.90	33.1	41.6	**27.6**	37.4	43.3
User Declaration	33.2	69.1	48.4	59.6	15.7	56.7	43.7
Quantized Bucketing - lv1	34.4	85.5	**56.3**	**62.4**	16.1	59.6	46.4
Quantized Bucketing - lv2	41.9	45.3	**56.3**	**62.4**	16.1	43.4	57.8
Quantized Bucketing - lv3	*N/A*	**91.0**	**56.3**	**62.4**	16.1	**93.9**	**71.3**
K-means Bucketing - lv1	34.4	85.5	**56.3**	**62.4**	16.1	59.6	46.4
K-means Bucketing - lv2	43.9	45.9	**56.3**	**62.4**	16.1	84.2	57.5
K-means Bucketing - lv3	*N/A*	**91.0**	**56.3**	**62.4**	16.1	**93.9**	**71.3**

Unit: percentage
(the higher the better)

Information about types of tasks leads to better performance!

Outline

- Workflow Systems for Productivity at Scale
- TaskVine: A Data Intensive Workflow System
  - Architecture
  - Data Handling
  - MiniTasks and Serverless
- Applications
- **Challenges and Future Work**

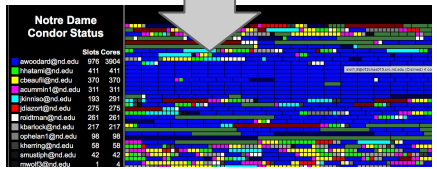
WIP: TaskVine and Dask

Ben Tovar

dask

```
# Dask Task Graph
d = {'x': 1,
     'y': (inc, 'x'),
       'z': (add, 'y', 10)}
```

TaskVine




```
if __name__ == "__main__":
    import dask.delayed

    @dask.delayed(pure=True)
    def sum_d(args):
        return sum(args)

    @dask.delayed(pure=True)
    def add_d(*args):
        return add(*args)

    @dask.delayed(pure=True)
    def list_d(*args):
        return list(args)

    z = add_d(1, 2)
    w = sum_d([1, 2, z])
    v = list_d(sum_d([z, w]), 2)
    t = sum_d(v)
```

```
m = DaskVine(port=0, ssl=True)

f = vine.Factory(manager=m)
f.cores = 4
f.max_workers = 1
f.min_workers = 1
with f:
    print(t.compute(scheduler=m.dask_get))
```

Open Problems

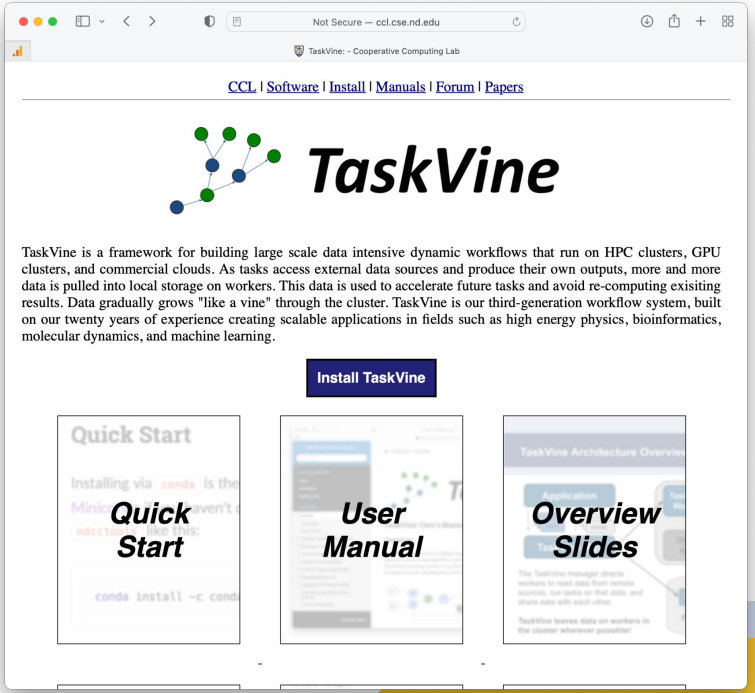
- **Adaptive Workload Decomposition:** How do we help users with the problem of "How big should my tasks be?" Too large: lost parallelism; too small: waste of overhead. In tension with:
- **Automated Resource Packing:** How do users decide the resources (cores, memory, disk) per task that achieve the best performance (throughput, utilization, runtime) for the users goals?
- **Understanding Dependencies:** How to help users disentangle what they want with what's installed with what was used last week…
- **Scheduling in Proportion:** There are countless techniques for scheduling workflows wrt dependencies, data, performance…
  But few scale up to millions of tasks on thousands of nodes!

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, Poncho, Resource Monitor, etc.
- First public release made in March 2023.
- 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



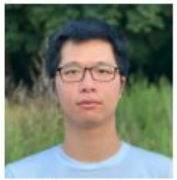
For more information…

This work was supported by NSF Award OAC-1931348

https://ccl.cse.nd.edu/software/taskvine
https://dthain.github.io

Douglas Thain
Director

Benjamin Tovar
Research Soft. Engineer



Thanh Son Phung
Ph.D. Student

Barry Sly Delgado
Ph.D. Student

Colin Thomas
Ph.D. Student

David Simonetti
Undergraduate

Joe Duggan
Undergraduate

Andrew Hennessee
Undergraduate

Matt Carbonaro
Undergraduate

Jachob Dolak
Undergraduate