Running Ensemble Workflows at Extreme Scale: Lessons Learned and Path Forward

IEEE eScience 2022

Kshitij Mehta[1], Ashley Cliff[2], Frédéric Suter[1], Angelica Walker[3], Matthew Wolf[1], Daniel Jacobson[1,3], Scott Klasky[1]

[1]Oak Ridge National Laboratory
[2]Lineberger Comprehensive Cancer Center, UNC
[3]University of Tennessee

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

Ensemble Workflows

- Ensemble Workflows – Executing multiple instances of a traditional workflow
  - Examples: Hyperparameter optimization in AI, simultaneous execution of multiple short-range simulations in MD codes
- Scaling ensemble workflows on a cluster/HPC system
  - Everyone focuses on efficient compute resource utilization
  - End goal is high task throughput
- Extreme scale execution more than just efficiently using CPUs

Use of WMS in HPC

- Limited adoption of WMS in HPC
- Application scientists hesitant to use WMS
- Too many WMS, lack of classification
- Over 300 systems listed here! →
- Limited support for WMS by HPC facilities
- Common practice to throw together a resource manager
- Will it scale to *extreme scale*?
- What should a WMS for extreme-scale science provide?

https://s.apache.org/existing-workflow-systems

Existing Workflow systems

Michael R. Crusoe edited this page 20 days ago · 329 revisions

Permalink: https://s.apache.org/existing-workflow-systems

Cite as (update dates):

Peter Amstutz, Maxim Mikheev, Michael R. Crusoe, Nebojša Tijanić, Samuel Lampa, et al. (2022): **Existing Workflow systems.** *Common Workflow Language wiki*, GitHub. https://s.apache.org/existing-workflow-systems updated 2022-08-30, accessed 2022-08-30.

Computational Data Analysis Workflow Systems

An incomplete list

Please add new entries at the bottom.

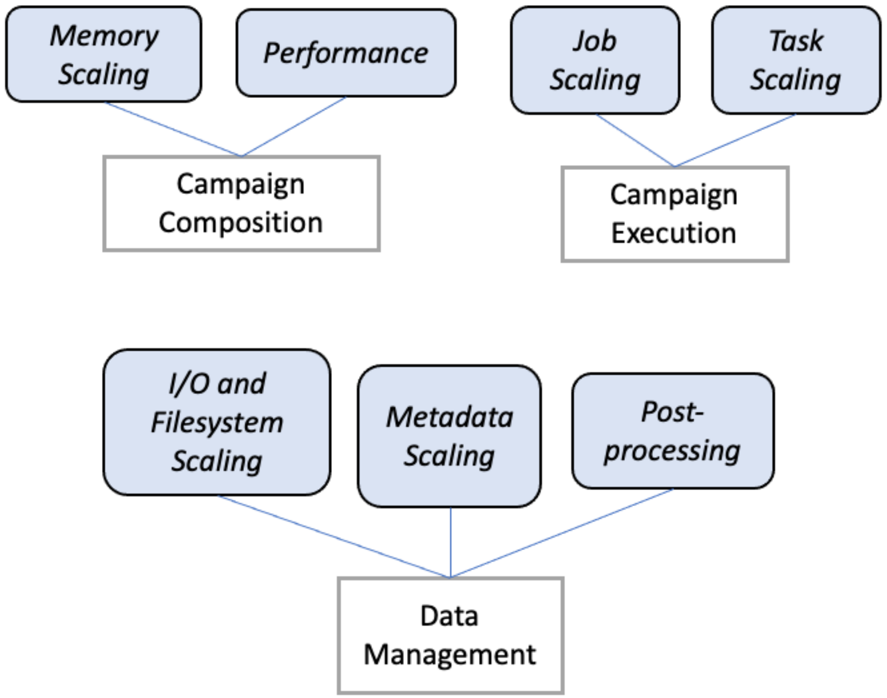
In addition to this list, actively developed free/open-source systems should be registered at https://workflows.community/systems

See also: https://github.com/pditommaso/awesome-pipeline

1. Arvados - CWL-based distributed computing platform for data analysis on massive data sets. https://arvados.org/ https://github.com/arvados/arvados
2. Apache Taverna http://www.taverna.org.uk/ https://taverna.incubator.apache.org/
3. Galaxy http://galaxyproject.org/
4. SHIWA https://www.shiwa-workflow.eu/
5. Apache Oozie https://oozie.apache.org/
6. DNANexus https://wiki.dnanexus.com/API-Specification-v1.0.0/IO-and-Run-Specifications https://wiki.dnanexus.com/API-Specification-v1.0.0/Workflows-and-Analyses
7. ~~BioDT http://www.biodatomics.com/~~ archived at https://web.archive.org/web/20180609011656/http://www.biodatomics.com/
8. Agave http://agaveapi.co/live-docs/
9. DiscoveryEnvironment http://www.iplantcollaborative.org/ci/discovery-environment
10. Wings http://www.wings-workflows.org/

Goal of this work

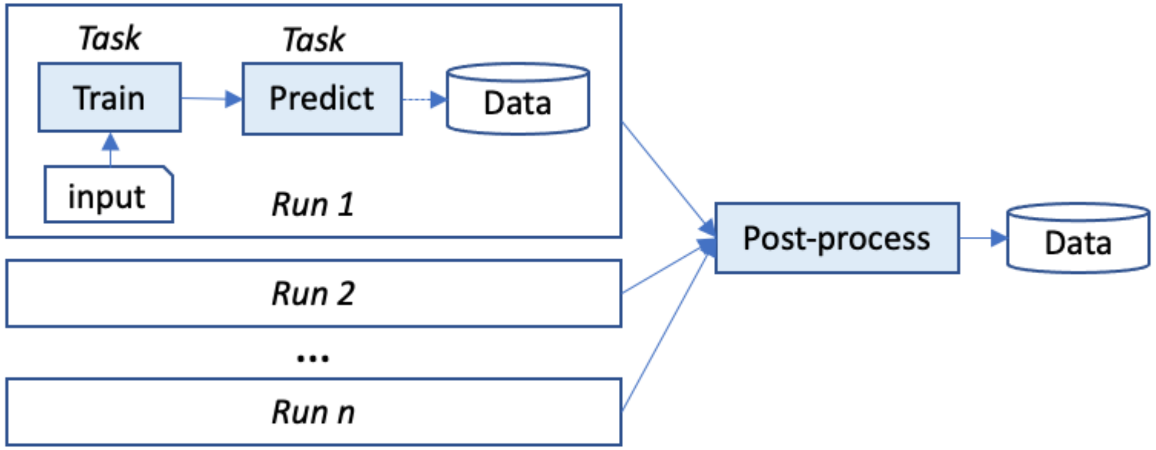
- Use a computational biology AI workflow for ensemble runs
- Use the Cheetah campaign management system from ECP for efficient resource management
- Highlight challenges encountered at extreme scale
  - Composition, execution, and data model
- Discuss lessons learned
- How to design an ensemble workflow from the ground up



A schematic of scaling challenges at extreme scale

iRF-LOOP

- The Iterative Random Forest Leave One Out Prediction (iRF-LOOP)
  - Iterative Random Forest algorithm for the creation of Predictive Expression Networks on the order of 40,000+ genes
- Multi-threaded C++ application
- Ensemble workflow runs a separate iRF instance for each *feature*
- Each instance generates *importance vector* files and model weights that are post-processed
- Each instance has its own workspace to avoid output filename collision



Oak Ridge National Laboratory

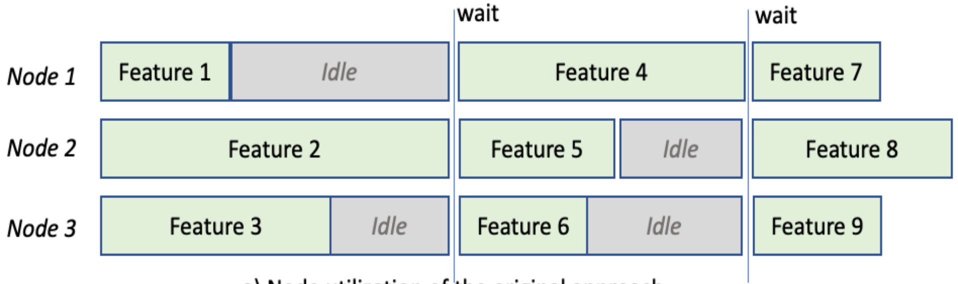
iRF-LOOP Execution – The Naïve Approach

- Scientists use shell scripts to run ensemble workflow
- Manually manage CPU resources

- Each instance runs on one node
- Submit a group of runs to fill nodes
- Wait statement acts as a synchronization barrier

- Severely underutilizes resources if different instances finish at different times
- Must use more sophisticated resource manager to dynamically spawn instances

```
#BSUB -nnodes 3

jsrun -p1 irfloop f1 &
jsrun -p1 irfloop f2 &
jsrun -p1 irfloop f3 &
wait

jsrun -p1 irfloop f6 &
jsrun -p1 irfloop f7 &
jsrun -p1 irfloop f8 &
wait
```



a) Node utilization of the original approach

Cheetah WMS

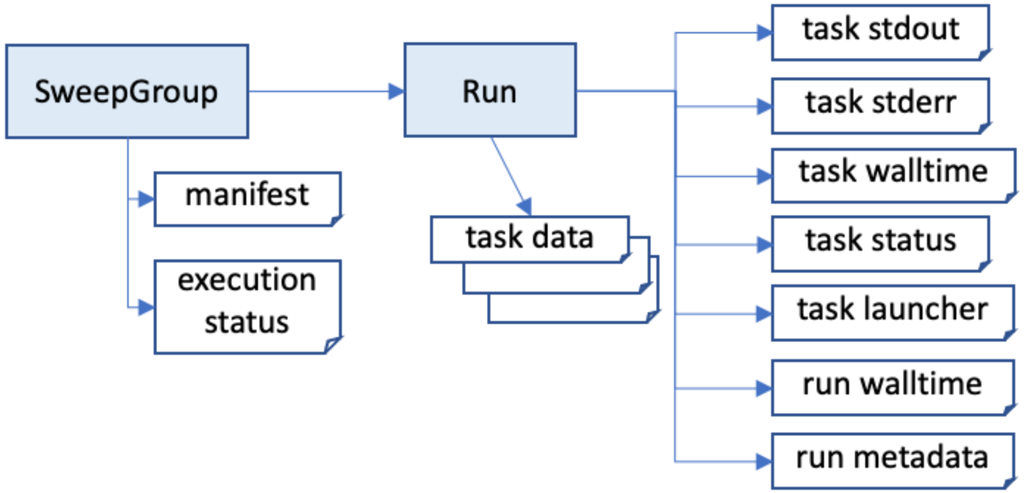
- WMS designed specifically for parameter sweeps
- Python-based API to create ‘Campaign’
- Runtime engine dynamically spawns instances over available resources

```
#BSUB -nnodes 3

jsrun -p1 irfloop f1 &
jsrun -p1 irfloop f2 &
jsrun -p1 irfloop f3 &
wait

jsrun -p1 irfloop f6 &
jsrun -p1 irfloop f7 &
jsrun -p1 irfloop f8 &
wait
```

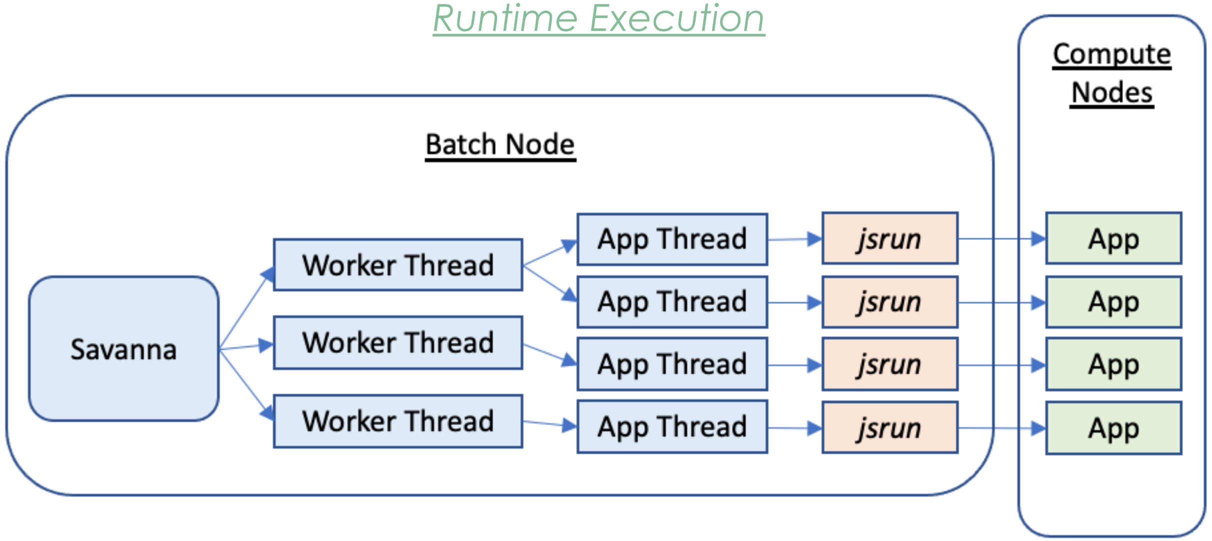
→

```
#BSUB -nnodes 3

jsrun -p1 cheetah
```

Campaign directory layout

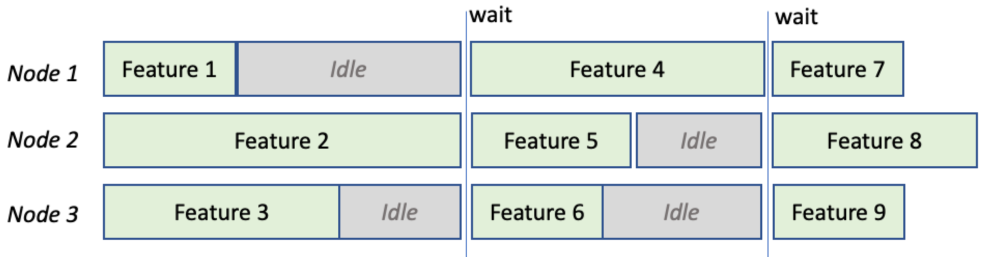


Runtime Execution

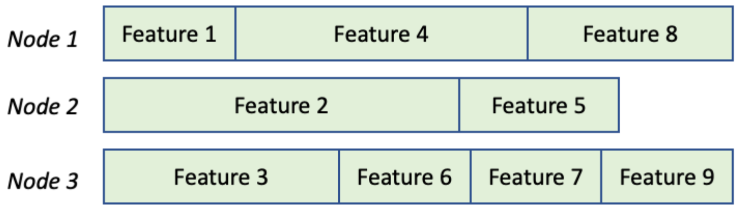


Executing the iRF-LOOP Ensemble using Cheetah

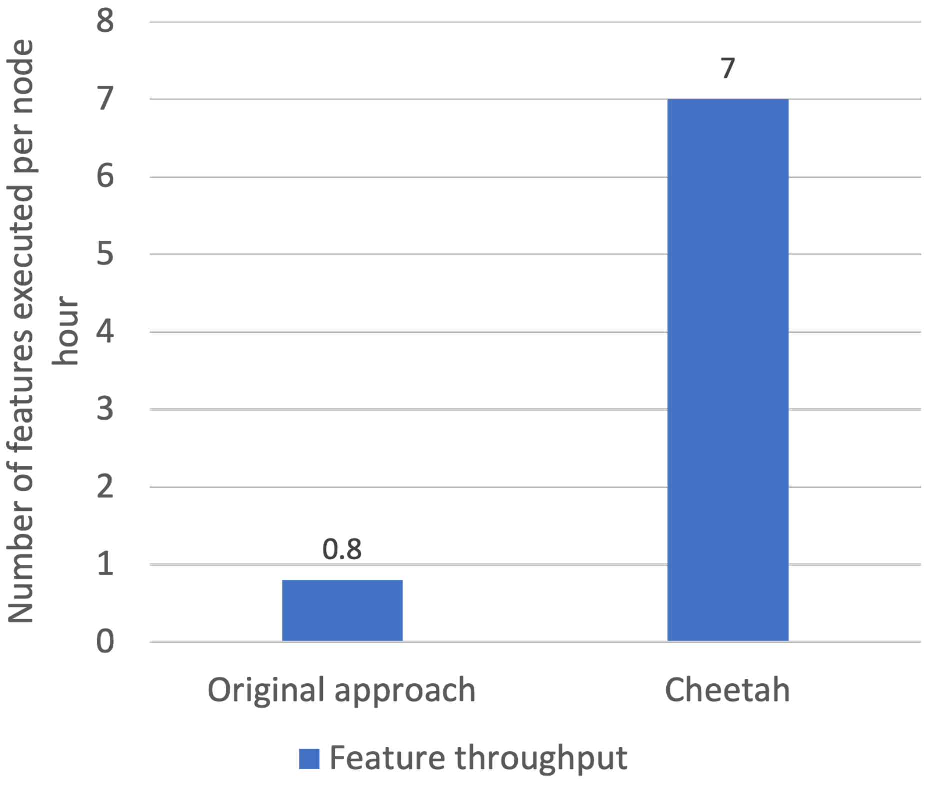
- Test ensemble using a community dataset consisting of 1,606 features
  - 1,606 * 50 test sets = 80,000+ runs



a) Node utilization of the original approach



b) Node utilization using Cheetah



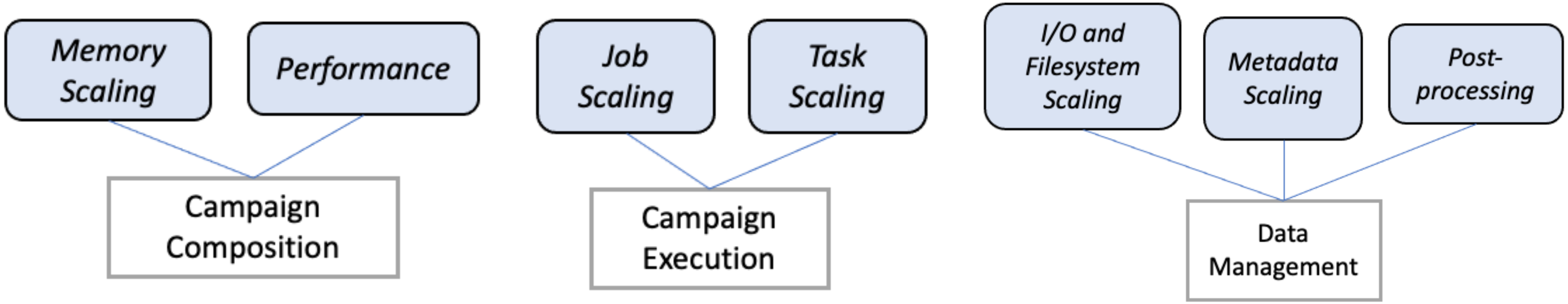
8.75x improvement in feature throughput

Ready to scale up?

- Using Cheetah easy
  - Short learning curve
  - Pure Python, easy to install
  - Good speedup with low effort 👍🏻
  - Fairly sophisticated WMS – create ensemble, execute, monitor, resume
- Lets scale up
- Process large dataset with 81,000 features
  - 81k features * 50 test sets = 4,00,000+ runs

- Two Campaign designs
- Capability class
  - One large batch job for all runs
  - Good for leadership-class supercomputers
  - Large resource allocation
- Capacity class
  - Large collection of batch jobs
  - Good for capacity-class supercomputers
  - Many, smaller resource allocations

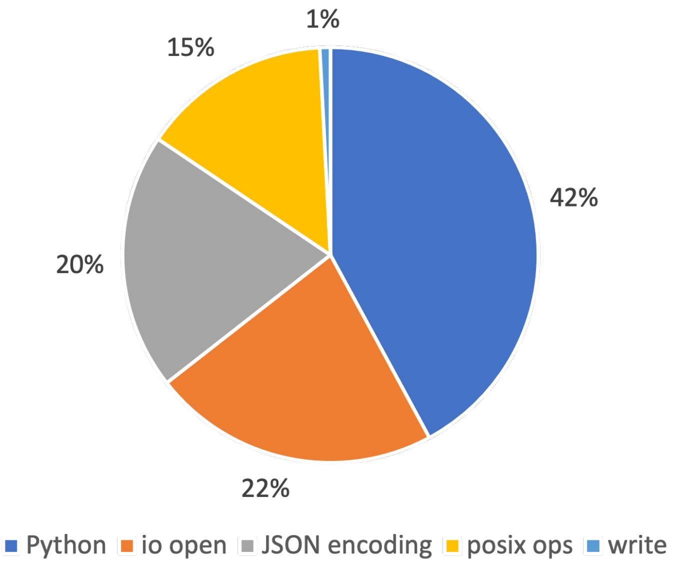
Challenges at Scale

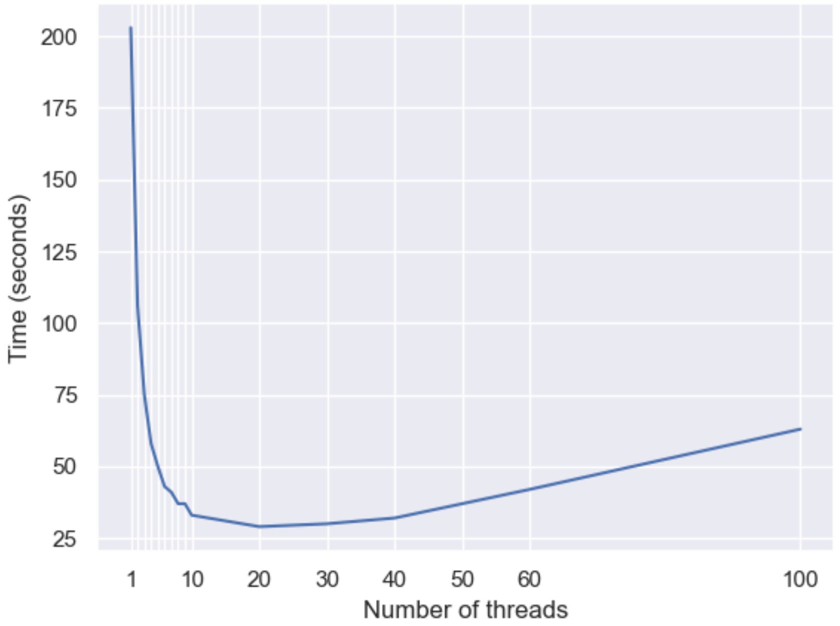


- Composing a large campaign - Memory and time
- Execution of a large campaign - Job scaling, task scaling
- Data scalability - I/O and file system, metadata

Lesson 1: Cost of Ensemble Setup

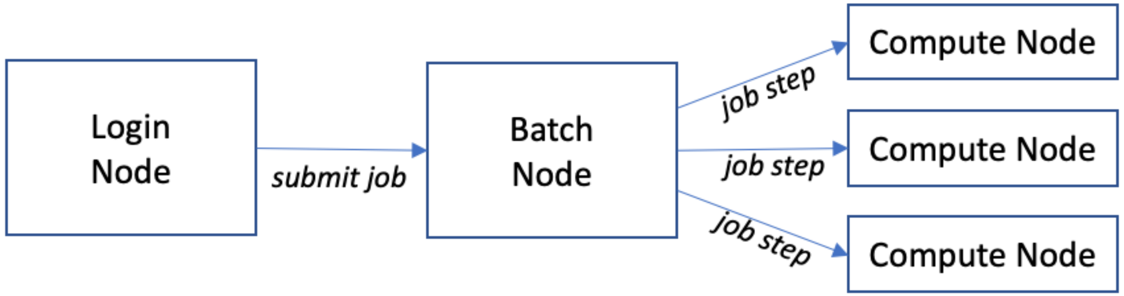
- Cost of composing the ensemble high
- Setup process runs out of memory for setting up large ensemble
  - Use Python generators to manage memory usage
- Almost 4 hours to create ensemble directories and files
  - 40% in file and dir operations, 20% in JSON operations, remaining 40% in Python processing
- ***Creating an ensemble directory hierarchy is memory intensive and time consuming at scale***



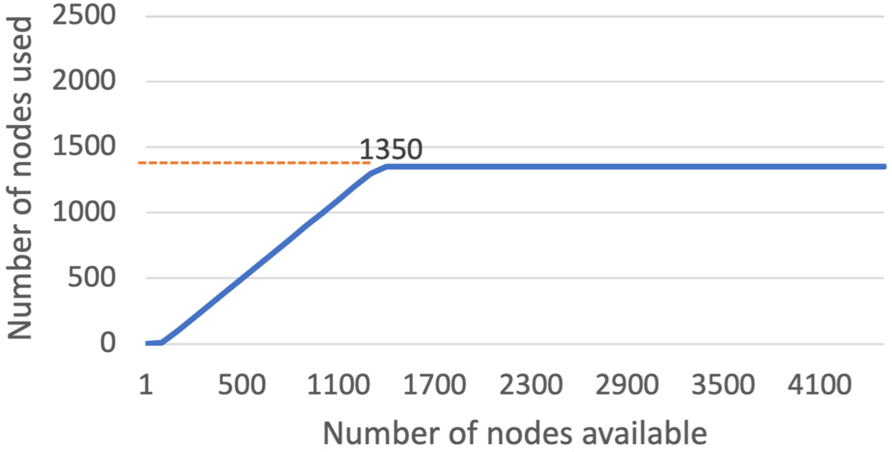


Lessons 2 and 3: Job and Task Scaling

- ***Lesson 2: Queue policies restrict the maximum number of jobs in queue***
- Limit of 5000 on Perlmutter, 100 on Summit
- Cannot submit full campaign of 80k jobs
- Solution is to use WMS with dynamic job management capabilities
  - HTCondor, Pegasus, Makeflow and more



- ***Lesson 3: Job Step Scalability limits Task Scalability***
- Job execution on most supercomputers: login node → batch node → compute nodes
- Limit on the no. of concurrent srun, jsrun job steps from a batch/service node
  - Max 1000+ jsrun invocations on Summit
- Limits no. of concurrent ensemble *runs*



Lesson 4: Limits on Metadata Scaling

- JSON/YAML formats common for metadata
  - Human readable, easy to use
- At extreme scale for iRF-LOOP, JSON metadata file over 50 MB!
- Updating a single value in JSON loads the entire file in memory
- Fast updates at scale lead to metadata bottleneck
- ***Popular file formats for metadata management perform poorly at scale***
- Need to switch over to *scalable* DB options

Lessons 5 and 6: Filesystem and Post-processing Overheads

- Files, files, and more files!
- Files provide an easy-to-use way to store data
- ***A few files per run easily leads to millions of files at scale***
- Filesystem scalability issues and limits on inode usage
- Post-processing – read back large no. of files for processing and analysis
- ***Post-processing data from a large ensemble is prohibitively expensive***
- A workflow consisting of a post-processing phase that reads back files bound to fail

Files created by the app in each instance	13
Files created by the WMS	12
No. of runs in the ensemble	> 4 million
No. of directories in the ensemble	> 4 million
Total no. of files expected	> 100 milllion

WMS for Extreme Scale Workflows: A Path Forward

- Dynamic Execution
  - Dynamically create batch jobs and assign runs to jobs
- Scalable Task Scheduling
  - Pilot-based systems and scalable resource management
- Strong integration with scientific data management
  - How to translate from a traditional file-based model to HDF5, ADIOS?
- Scalable metadata management
  - Export API for metadata storage
- Automatic provisioning of storage hierarchy
  - Transparently use tiered storage
- Online data analysis
  - Abstractions to easily move from post-processing to in situ

Summary

- Challenges in scaling ensemble workflows to extreme scale
- Initial application design must include efficient data management
  - Cannot liberally use files for data and metadata
  - File-based post-processing workflow cannot scale
- WMS must include scalable job and task scheduling
  - Easy integration into an existing workflow
- Easily integrate hardware resources such as tiered storage
- How to bring together strengths and features of different WMS

Thank you