Balsam Workflows and Ensemble Jobs

Misha Salim
ALCF Data Science Group

msalim@anl.gov
Overview

- When can Balsam be useful?
- What is Balsam?
- Starting on Theta
  - Case study: managing a large campaign of simulations
  - Data dependencies and dynamic workflows
  - Case study: hyperparameter optimization
Overview

• When can Balsam be useful?

• What is Balsam?

• Starting on Theta

• Case study: managing a large campaign of simulations

• Data dependencies and dynamic workflows

• Case study: hyperparameter optimization
What is a workflow?

- Strategy for executing your application runs
  - Job scheduling
  - “Packing” small runs into larger ensemble jobs
- Data movement
- Pre- and post-processing

In some cases, a few command line tools is enough
(20 jobs) (1024 nodes) (12 hours) = 245,760 node-hours

Other cases demand a more careful approach
(2,457,600 jobs) (1 node) (6 minutes) = 245,760 node-hours
Theta Ensemble Jobs: Quick Solution

- Up to 1000 concurrent apruns per Cobalt job
- Easy way to achieve throughput, given the ALCF queueing policies

```bash
-----myjob.sh------
#!/bin/sh
echo "Starting Cobalt job script"
aprun -n 128 -N 64 run1.exe arg1 &
sleep 1
aprun -n 256 -N 64 run1.exe arg1 &
sleep 1
aprun -n 512 -N 64 run1.exe arg1 &
wait
-----end myjob.sh-----
```
This approach needs help to scale

- Large number of application runs
- Dependencies in workflow
- Maintaining high resource utilization
- Tracking finished/failed/timed-out runs, modifying inputs, and dispatching new batches of work becomes cumbersome

Write your own scripts? Invest in a workflow manager?
Why workflow management?

- Organize large compute campaigns
- Automate scheduling & job dispatch
- Maximize concurrency to exploit LCF resources
- Improve robustness with error-handling and retry capabilities
- Record provenance data for:
  - Faster detection of errors
  - Workflow execution statistics (e.g. throughput over time)
Overview

• When can Balsam be useful?

• What is Balsam?

• Starting on Theta

• Case study: managing a large campaign of simulations

• Data dependencies and dynamic workflows

• Case study: hyperparameter optimization
Balsam components

Service

DB

Launcher

HPC Scheduler

Login node

Head node

Backend

mpirun

qsub
Balsam components

Database

BalsamJob table stores workflow state

One row for each planned application run
Balsam components

**Launcher**

Pilot application running inside Cobalt job

Dynamic task pull and execution
Balsam components

Service

Elastic scheduling of launcher jobs under varying workload
Balsam Highlights

- No modification of user applications
- Command line and Python interfaces
- Performance tested up to 1.2M jobs in database
- Strengths & limitations of launcher on Theta are well-understood
- Strong task-level fault tolerance
  - even fatal errors do not interrupt workflow
- Concurrent execution of serial and MPI applications across one or many Cobalt jobs
Other approaches

Cram runs many small MPI jobs inside one large MPI job.

- [https://github.com/llnl/cram](https://github.com/llnl/cram)
- SPMD approach: link your application against libcram
- Create a “CramFile” with args for each application instance
  - Unlimited MPI jobs can be packed into one; no strain on head node
  - All instances start concurrently; job takes as long as the the slowest instance
  - All tasks run under MPI_COMM_WORLD; one fault can abort the entire run
Other approaches

- Express workflow as code with dataflow constructs
- Tasks eligible for concurrent execution automatically distributed
- Concise and flexible language for defining workflows
- Swift/T backend provides high performance, scalable MPI runtime environment for workflow
- Running “MPI-inside-MPI” becomes a technical hurdle
- Lack of resource isolation between Swift/T tasks hampers fault tolerance
- No notion of centralized database; may be less appropriate for extended computational campaigns
Overview

• When can Balsam be useful?
• What is Balsam?

• Starting on Theta
  • Case study: managing a large campaign of simulations
  • Data dependencies and dynamic workflows
  • Case study: hyperparameter optimization
Balsam on Theta — Walkthrough

- [https://balsam.alcf.anl.gov/index.html](https://balsam.alcf.anl.gov/index.html) — documentation undergoing major update

- Up-to-date module available on Theta:

  msalim@thetalogin6:~> module load balsam
  msalim@thetalogin6:~> balsam
  Balsam requires Python version $>=$ 3.6
Command line interface

msalim@thetalogin6:~> module load cray-python/3.6.1.1
msalim@thetalogin6:~> balsam
usage: balsam [-h]
    {app,job,dep,ls,modify,rm,killjob,mkchild,launcher,submit-launch,
     mmies,which,log,server}
...

Balsam command line interface

optional arguments:
    -h, --help       show this help message and exit

Subcommands:
    {app,job,dep,ls,modify,rm,killjob,mkchild,launcher,submit-launch,init,service,
     log,server}

    app            add a new application definition
    job            add a new Balsam job
    dep            add a dependency between two existing jobs
    ls             list jobs, applications, or jobs-by-workflow
    modify         alter job or application
    rm              remove jobs or applications from the database
    killjob        Kill a job without removing it from the DB
    mkchild        Create a child job of a specified job
    launcher       Start a local instance of the balsam launcher
Start a new Balsam DB

Use `balsam init` to create a new database directory

```bash
msalim@thetalogin6:~> balsam init ~/test-db
```

Successfully created Balsam DB at: `/home/msalim/test-db`
Use `source balsamactivate test-db` to begin working.
Start a new Balsam DB

source balsamactivate <db-name> : starts server, if not already running, sets environment for Balsam

msalim@thetalogin6:~> . balsamactivate test-db
Launching Balsam DB server
waiting for server to start.... done
server started
[BalsamDB: test-db] msalim@thetalogin6:~>
Hello World

balsam app:
Register new applications with Balsam

[BalsamDB: test-db] msalim@thetalogin6:~> balsam app --name say-hello --executable "echo Hello,"
Application 1:
-------------------------------
Name: say-hello
Description: 
Executable: echo Hello,
Preprocess: 
Postprocess: 

Added app to database
Hello World

balsam job:
Add a new run (or task) to the database

[BalsamDB: test-db] msalim@thetalogin6:~> balsam job --name test1 --workflow test
> --app say-hello --args "world!"
Hello World

Confirmation shows task details and names of adjustable fields

- workflow: test
- name: test1
- description: 
- lock: []
- parents: [*]
- input_files: 
- stage_in_url: 
- stage_out_files: 
- stage_out_url: 
- wall_time_minutes: 1
- num_nodes: 1
- coschedule_num_nodes: 0
- ranks_per_node: 1
- cpu_affinity: none
- threads_per_rank: 1
- threads_per_core: 1
- node_packing_count: 1
- environ_vars: 
- application: say-hello
- args: 
- user_workdir: 
- wait_for_parents: True
- post_error_handler: False
- post_timeout_handler: False
Hello World

Confirmation shows task details and names of adjustable fields

auto_timeout_retry: True
state: CREATED
queued_launch_id: None
data: {}

*** Executed command: echo Hello, world!
*** Working directory: /gpfs/mira-home/msalim/test-db/data/test/test1_7df86c8b
Hello World

Adding 9 more jobs...

[BalsamDB: test-db] msalim@thetalogin6:~> for i in {2..10} 
> do 
> balsam job --name test${i} --workflow test --app say-hello \ 
> --args "world ${i}!" --yes 
> done
Hello World

**balsam ls:**

**View tasks in database**

[BalsamDB: test-db] msalim@thetalogin6:~> balsam ls

<table>
<thead>
<tr>
<th>job_id</th>
<th>name</th>
<th>workflow</th>
<th>application</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>7df86c8b-f94b-4349-8ab6-a8afc85d8a9f</td>
<td>test1</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
<tr>
<td>ab58c816-0665-4cd3-b3f2-e16312e77887</td>
<td>test2</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
<tr>
<td>ff23e57b-6829-4abc-966d-82e2c6a90d9e</td>
<td>test3</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
<tr>
<td>5bee0827-99bb-4b8b-ac7a-a7dd15e4bda3</td>
<td>test4</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
<tr>
<td>d4705a0a-bf6d-4f7e-a4fb-146a9bfe55a1</td>
<td>test5</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
<tr>
<td>b830798e-a4f8-45b1-b7bd-dd62adb92ac</td>
<td>test6</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
<tr>
<td>51e23e81-0655-436e-a1f0-550b6a1b3102</td>
<td>test7</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
<tr>
<td>6a87c9cf-19d5-4796-8d3b-1c6115067ad5</td>
<td>test8</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
<tr>
<td>cac08656-8056-452f-b709-bdadd21ccf46</td>
<td>test9</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
<tr>
<td>14f94e50-9076-4db5-ac3f-de7977717b8c</td>
<td>test10</td>
<td>test</td>
<td>say-hello</td>
<td>CREATED</td>
</tr>
</tbody>
</table>
Hello World

balsam submit-launch:
Shortcut for Cobalt job submission (template in ~/.balsam)

[BalsamDB: test-db] msalim@thetalogin6:~> balsam submit-launch -n 2 -t 5 -q debug-cache-quad -A SDL_Workshop -q training --job-mode=serial
Submit OK: Qlaunch {  'command': '/gpfs/mira-home/msalim/test-db/qsubmit/qlaunch1.sh',
   'from_balsam': True,
   'id': 1,
   'job_mode': 'serial',
   'nodes': 2,
   'prescheduled_only': False,
   'project': 'SDL_Workshop',
   'queue': 'training',
   'scheduler_id': 278079,
   'state': 'submitted',
   'wall_minutes': 5,
   'wf_filter': ''}
Hello World

If successful, jobs eventually marked
JOB_FINISHED

[BalsamDB: test-db] msalim@thetalogin6:~/test-db/qsubmit> balsam ls

job_id  name  workflow  application  state

| cac08656-8056-452f-b709-bdadd21ccf46 | test9 | test | say-hello | JOB_FINISHED |
| b830798e-a4f8-45b1-b7bd-dd62adfb92ac | test6 | test | say-hello | JOB_FINISHED |
| ff23e57b-6829-4abc-966d-82e2c6a90d9e | test3 | test | say-hello | JOB_FINISHED |
| ab58c816-0665-4cd3-b3f2-e16312e77887 | test2 | test | say-hello | JOB_FINISHED |
| 7df86c8b-f94b-4349-8ab6-a8afc85d8a9f | test1 | test | say-hello | JOB_FINISHED |
| 6a87c9cf-19d5-4796-8d3b-1c6115067ad5 | test8 | test | say-hello | JOB_FINISHED |
| 51e23e81-0655-436e-a1f0-550b6a1b3102 | test7 | test | say-hello | JOB_FINISHED |
| 14f94e50-9076-4db5-ac3f-de7977717b8c | test10 | test | say-hello | JOB_FINISHED |
| d4705a0a-bf6d-4f7e-a4fb-146a9bfe55a1 | test5 | test | say-hello | JOB_FINISHED |
| 5bee0827-99bb-4bb8-ac7a-a7dd15e4bda3 | test4 | test | say-hello | JOB_FINISHED |
Hello World

balsam ls --hist:
View state history metadata

[BalsamDB: test-db] msalim@thetalogin6:~/test-db/qsubmit> balsam ls --name test5 --history
Job test5 [d4705a0a-bf6d-4f7e-a4fb-146a9bfe55a1]

[10-03-2018 19:04:00.143928 CREATED]
Where did the output go?

By default, everything goes into DB directory

```
[BalsamDB: test-db] msalim@thetalogin6:~:/test-db> ls
balsamdb  data  log  qsubmit  server-info
```

Job working directories are created as:

```
data/<workflow>/<name>_<id>
```

```
[BalsamDB: test-db] msalim@thetalogin6:~:/test-db> ls data/test/
test10_14f94e50  test2_ab58c816  test4_5bee0827  test6_b830798e  test8_6a87c9cf  testfail_fab575a3
test1_7df86c8b  test3_ff23e57b  test5_d4705a0a  test7_51e23e81  test9_cac08656
```
Error States

Balsam handles failed runs gracefully

```python
from mpi4py import MPI

rank = MPI.COMM_WORLD.Get_rank()
if rank == 0:
    raise RuntimeError("simulated error")
else:
    print("Hello from rank", rank)
```
Error States

[BalsamDB: test-db] msalim@thetalogin6:/test-db/data/test/testfail_fab575a3> balsam ls

<table>
<thead>
<tr>
<th>job_id</th>
<th>name</th>
<th>workflow</th>
<th>application</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>fab575a3-01db-41b5-b70d-c396c17ef10d</td>
<td>testfail</td>
<td>test</td>
<td>failer</td>
<td>FAILED</td>
</tr>
<tr>
<td>cac08656-8056-452f-b709-bdadd21ccf46</td>
<td>test9</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
<tr>
<td>b830798e-a4f8-45b1-b7bd-dd62adfb92ac</td>
<td>test6</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
<tr>
<td>ff23e57b-6829-4abc-966d-82e2c6a90d9e</td>
<td>test3</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
<tr>
<td>ab58c816-0665-4cd3-b3f2-e16312e77887</td>
<td>test2</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
<tr>
<td>7df86c8b-f94b-4349-8ab6-a8afc85d8a9f</td>
<td>test1</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
<tr>
<td>6a87c9cf-19d5-4796-8d3b-1c6115067ad5</td>
<td>test8</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
<tr>
<td>51e23e81-0655-436e-a1f0-550b6a1b3102</td>
<td>test7</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
<tr>
<td>14f94e50-9076-4db5-ac3f-de7977717b8c</td>
<td>test10</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
<tr>
<td>d4705a0a-bf6d-4f7e-a4fb-146a9bfe55a1</td>
<td>test5</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
<tr>
<td>5bee0827-99bb-4bbb-ac7a-a7dd15e4bda3</td>
<td>test4</td>
<td>test</td>
<td>say-hello</td>
<td>JOB_FINISHED</td>
</tr>
</tbody>
</table>
Error States

balsam ls --state:
Filter jobs by state

[balsam DB: test-db] msalim@thetalogin6:/test-db/data/test/testfail_fab575a3> balsam ls --state FAILED --history
Job testfail [fab575a3-01db-41b5-b70d-c396c17ef10d]

[10-03-2018 19:34:38.379895 CREATED]
  Hello from rank 2
  Hello from rank 1
  File "/gpfsmira-home/msalim/test-db/qsubmit/fail.py", line 5, in <module>
    raise RuntimeError("simulated error")
RuntimeError: simulated error
Hello from rank 4
Hello from rank 3
Hello from rank 5
Application 5762994 exit codes: 1
Application 5762994 resources: utime ~2s, stime ~4s, Rss ~19956, inblocks ~22664, outblocks ~0
[10-03-2018 19:38:34.516193 FAILED]
Modifying Tasks

Modify BalsamJob fields from command line:

```
balsam modify fab5 state RESTART_READY
```

Or with more flexible Python API:

```python
>>> from balsam.launcher.dag import BalsamJob
>>> BalsamJob.objects.filter(num_nodes__lte=128, name__contains="test", state="JOB_FINISHED").update(
... state="RESTART_READY")
```
Overview

• When can Balsam be useful?
• What is Balsam?
• Starting on Theta

• **Case study: managing a large campaign of simulations**
• Data dependencies and dynamic workflows
• Case study: hyperparameter optimization
Populating the Database

• “Constructing and Navigating Polymorphic Landscapes of Molecular Crystals” ADSP (PI: Alexandre Tkatchenko)

• High-throughput characterization of ~1.2M structures by DFT

• How do we get all of these jobs into Balsam?
for (dirpath, dirnames, filenames) in os.walk(inbox_path):
    xyz_files = [f for f in filenames if f.endswith('.xyz')]
    new_jobs = []
    for f in xyz_files:
        name = os.path.splitext(f)[0]
        workflow = os.path.basename(dirpath)
        xyz_path = os.path.join(dirpath, f)
        job = prep_job(name, workflow, xyz_path)
        new_jobs.append(job)
    BalsamJob.objects.bulk_create(new_jobs)
print("Created", len(new_jobs), "new jobs in DB")
def prep_job(name, workflow, xyz_path):
    return BalsamJob(name=name,
                     workflow = workflow,
                     stage_in_url = xyz_path,
                     application = 'fhi-aims',
                     ranks_per_node=64,
                     cpu_affinity='depth',
                     environ_vars="OMP_NUM_THREADS=1",
                     )
Submitting Jobs

• Use “submit-launch” shortcut to enqueue ~10-20 large jobs
• Each launcher job consumes as much available work as it can
• Concurrent launcher jobs work cooperatively by means of application-enforced locks acquired in the database

• Alternatively, Balsam service can manage job packing and flow into scheduler
The Launcher job template

```
"SCHEDULER_CLASS": "CobaltScheduler",
"SCHEDULER_SUBMIT_EXE": "/usr/bin/qsub",
"SCHEDULER_STATUS_EXE": "/usr/bin/qstat",
"DEFAULT_PROJECT": "datascience",
"SERVICE_PERIOD": 1,

"NUM_TRANSITION_THREADS": 5,
"MAX_CONCURRENT_MPIRUNS": 1000,

"LOG_HANDLER_LEVEL": "INFO",
"LOG_BACKUP_COUNT": 5,
"LOG_FILE_SIZE_LIMIT": 104857600,

"QUEUE_POLICY": "theta_policy.ini",
"JOB_TEMPLATE": "job-templates/theta.cobaltscheduler.tmpl"
```
The Launcher job template

```bash
#!/bin/bash -x
#COBALT -A {{ project }}
#COBALT -n {{ nodes }}
#COBALT -q {{ queue }}
#COBALT -t {{ time_minutes }}
#COBALT --attrs ssds=required:ssd_size=128

...

source balsamactivate {{ balsam_db_path }}
sleep 2

balsam launcher --{{ wf_filter }} --job-mode=\{\{ job_mode \}\} --time-limit-minutes=\{\{ time_minutes-2 \}\}
source balsamdeactivate
```
Configurable Queue Submission Policy

```yaml
[debug-flat-quad]
submit-jobs = on
max-queued = 1
policy = [
  {
    "min-nodes": 1,
    "max-nodes": 16,
    "min-time": 0,
    "max-time": 1
  }
]
```
Monitoring Progress

balsam ls --by-states:
Group by state
## Monitoring Progress

<table>
<thead>
<tr>
<th>JOB_FINISHED (573989 BalsamJobs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_id</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>7ccbeb07b-ea1b-4567-8dbc-6d0b4f8ba51e</td>
</tr>
<tr>
<td>02b806d7-ddf4-42e8-9367-e667651da55</td>
</tr>
<tr>
<td>(573993 more BalsamJobs not shown...)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PREPROCESSED (72409 BalsamJobs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_id</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>91725337-6226-4302-8dddb-f97044621b1a</td>
</tr>
<tr>
<td>91827748-7550-41a4-bbd9-88ac1d2b131d</td>
</tr>
<tr>
<td>(72402 more BalsamJobs not shown...)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESTART_READY (5396 BalsamJobs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_id</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>2c03e5cd-2c94-4376-8c1c-5cd8abb6adbf</td>
</tr>
<tr>
<td>2c0325ac-3135-424c-b332-1790e4e6c13</td>
</tr>
<tr>
<td>(5394 more BalsamJobs not shown...)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RUN_DONE (17 BalsamJobs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_id</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>34020491-3c31-4dc2-a9a4-5aa622fc517d</td>
</tr>
<tr>
<td>251ac1f1-566f-446b-9245-0a1079a7010f</td>
</tr>
<tr>
<td>(10 more BalsamJobs not shown...)</td>
</tr>
</tbody>
</table>

| RUNNING (797 BalsamJobs) |
Overview

• When can Balsam be useful?
• What is Balsam?
• Starting on Theta
• Case study: managing a large campaign of simulations
• Data dependencies and dynamic workflows
• Case study: hyperparameter optimization
from balsam.launcher.dag import (add_job, add_dependency)

A = add_job(name="A", application="generate")
B,C,D = [
    add_job(
        name=name,
        application="simulate",
        input_files=name+'.inp'
    )
    for name in "BCD"
]
E = add_job(name="E", application="reduce", input_files="*.out")

for job in B,C,D:
    add_dependency(A, job)
    add_dependency(job,E)
Dynamic Workflows

• `balsam.launcher.dag` enables context-aware processing
  - `dag.current_job`

• Attach pre/post-processing scripts to application
  - **Inspect** `dag.current_job`
  - Modify DB in response to workflow outcomes

• `dag.kill(job)`
  - terminate and replace launcher tasks in near-realtime
Dynamic Workflows

• In more tightly-coupled workflows, a “master” app running under Balsam may itself spawn tasks for parallel, asynchronous execution

• `balsam.launcher.async` facilitates programmatic polling/processing of BalsamJobs
  • borrow heavily from `concurrent.futures` API
  • reduces DB polling logic in user code
from balsam.launcher.dag import add_job
from balsam.launcher.async import wait, FutureTask

def on_done(job):
    return job.read_file_in_workdir(f'\{job.name}.out')

futures = []
for i in range(10):
    j = add_job(name=f'task{i}',
                application='say-hello',
                args=f'world {i}!'
              )
    futures.append(FutureTask(j, on_done))

results = wait(futures, return_when='ANY_COMPLETED', timeout=30)
Overview

• When can Balsam be useful?
• What is Balsam?
• Starting on Theta
• Case study: managing a large campaign of simulations
• Data dependencies and dynamic workflows

• Case study: hyperparameter optimization
DeepHyper: Hyperparameter Optimization

- **Prasanna Balaprakash (MCS)**
- Framework for defining hyperparameter search problems
  - Repository has ~20 ML benchmarks and growing…
- Variety of generic optimization methods
- Optimizers use **Evaluator abstraction** for asynchronous model training/validation with Balsam
- Theta release forthcoming…
DeepHyper+Balsam: Throughput on Theta

- Random Search over 8 h.p.s
- rnn2 (question-answering) model benchmark
- 96% weak-scaling efficiency from 128 to 1024 nodes
Utilization Profiles from Balsam

- BalsamJob metadata from a Bayesian surrogate model-based search run on 64 Cooley nodes
- Balsam provides convenience functions to parse state history and generate throughput/utilization profiles
Summary

• Balsam is under active development and in rapidly growing use for ADSP projects and beyond

• Facilitates large campaigns of ensemble jobs, with task-level fault tolerance and persistent provenance data

• Tools for automatic scheduling and dynamic workflows facilitate complex use cases