Using Balsam for Ensemble Workflows

Misha Salim

Argonne NATIONAL LABORATORY

Overview

- Balsam components & job database
- Balsam “Hello World”
- Hyperparameter optimization: Balsam usage

Ensemble Job Management

- Manage job DB via command-line interface or Python API
- Services for data transfer, job scheduling, pre- and post-processing, automatic ensemble execution
- Develop workflows with job dependencies and dynamic job creation/termination

Balsam Setup

- https://xgitlab.cels.anl.gov/datascience/balsam
- Requires Python 3.6 & mpi4py

balsam

0.1

Search docs

QUICKSTART

Install Balsam

Prerequisites

Environment

Get Balsam

Quick Tests

The Balsam Database

Hello World and Testing

USER GUIDE

Overview

Docs » Install Balsam

View page source

Install Balsam

Note

If you are reading this documentation from GitHub/GitLab, some of the example code is not displayed! Take a second and build this documentation on your own machine (until it's hosted somewhere accessible from the internet):

```
$ pip install --user sphinx
$ cd docs
$ make html
$ firefox _build/html/index.html # or navigate to file from browser
```

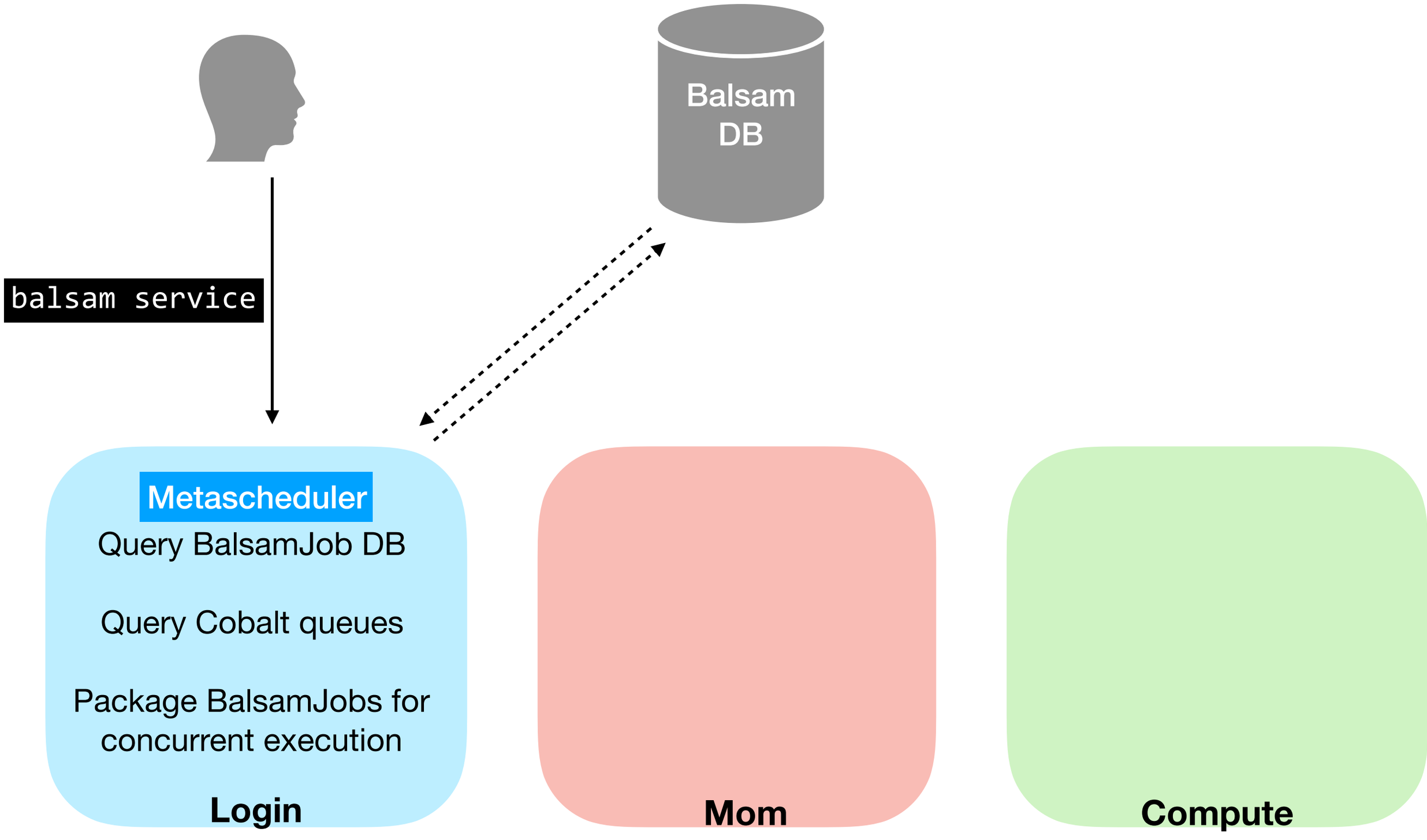
Balsam Components

Login

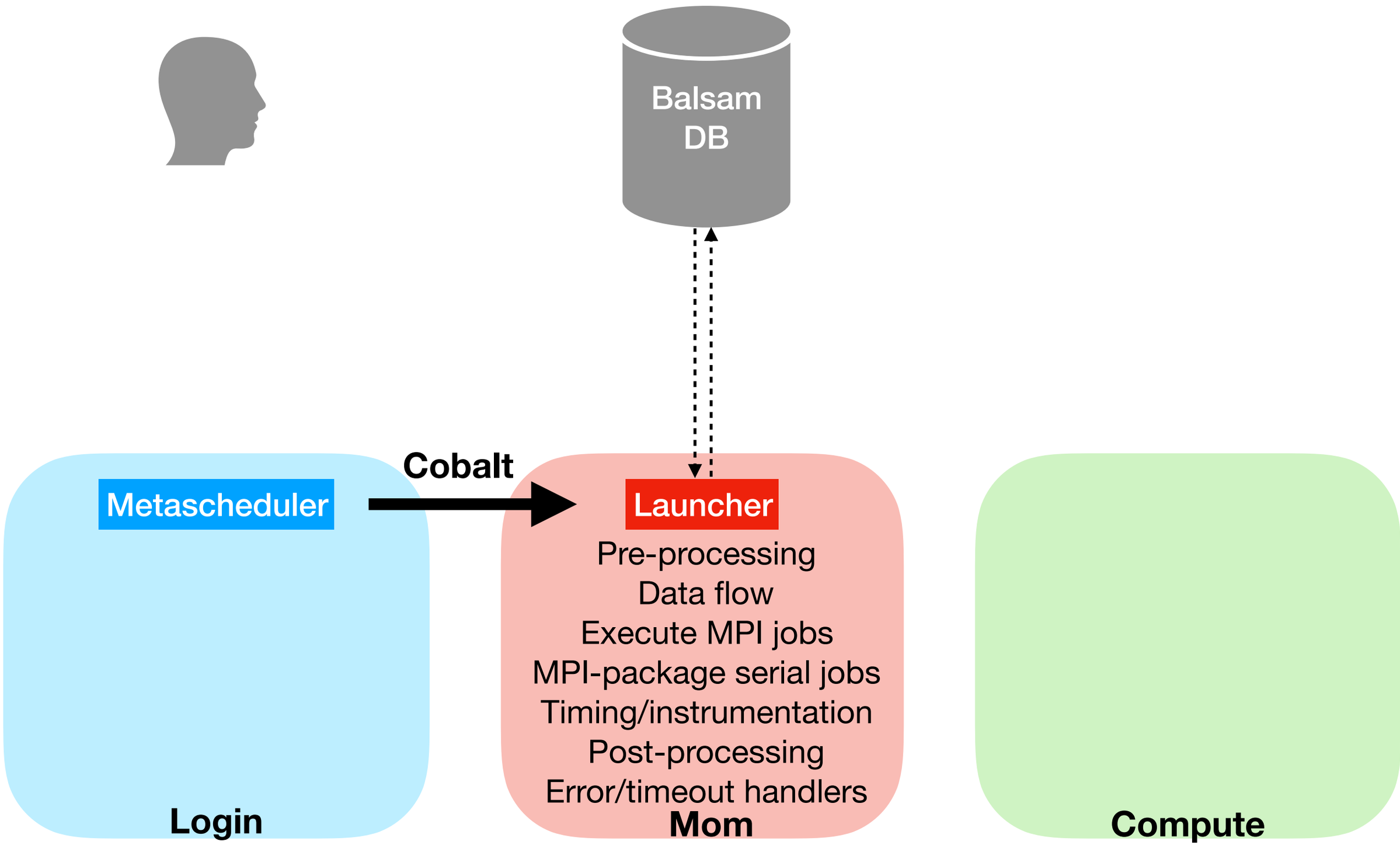
Mom

Compute

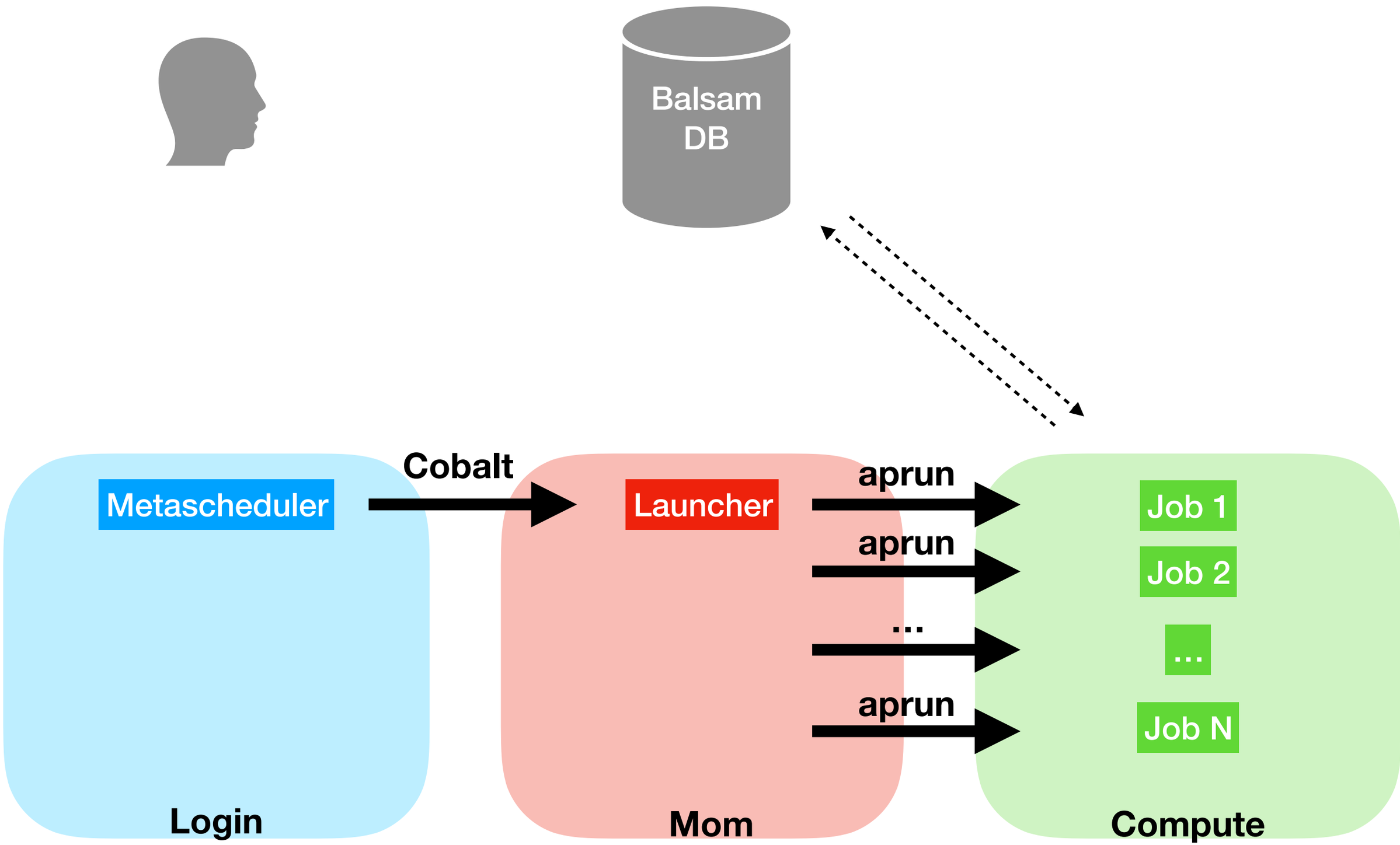
Balsam Components



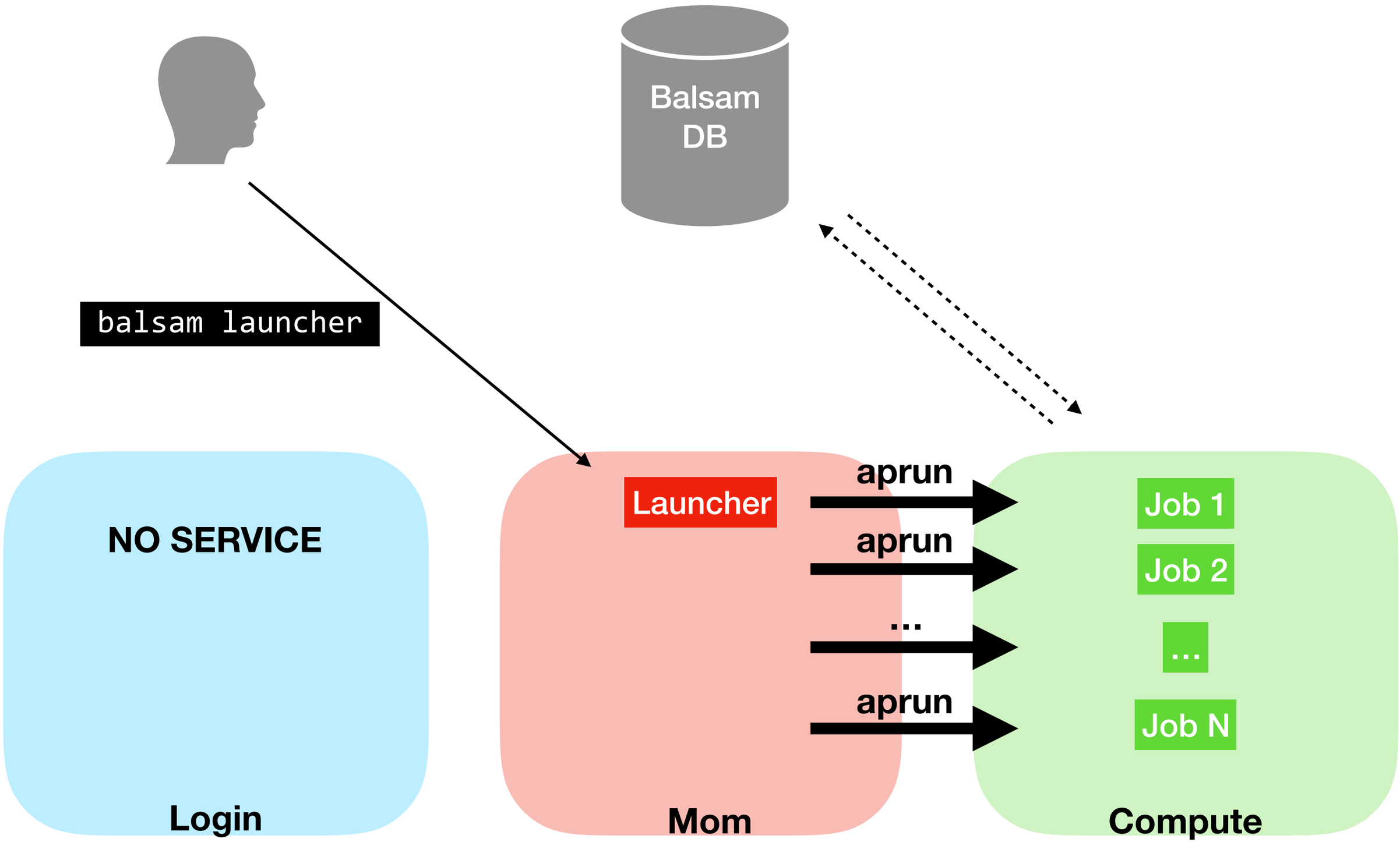
Balsam Components



Balsam Components



Balsam Components



BalsamJob Fields

ID / provenance

Unique ID
Name
Workflow name
Description
State history

Data flow

Parent Jobs
Working directory
Input file patterns
Stage-in file patterns / URL
Stage-out file patterns / URL

MPI / app args

Walltime estimate
Measured run time
nodes
ranks / node
hyperthreads / rank
hyperthreads / core
environment variables
command-line arguments

Executable Info

application name
(points to registered executable)

preprocess script
postprocess script
post: error handle flag
post: timeout handle flag
auto timeout retry count

BalsamJob Fields

ID / provenance

Unique ID
Name
Workflow name
Description
State history

Data flow

Parent Jobs
Working directory
Input file patterns
Stage-in file patterns / URL
Stage-out file patterns / URL

MPI / app args

Walltime estimate
Measured run time
nodes
ranks / node
hyperthreads / rank
hyperthreads / core
environment variables
command-line arguments

Executable Info

application name
(points to registered executable)

preprocess script
postprocess script
post: error handle flag
post: timeout handle flag
auto timeout retry count

BalsamJob Fields

ID / provenance

Unique ID
Name
Workflow name
Description
State history

Data flow

Parent Jobs
Working directory
Input file patterns
Stage-in file patterns / URL
Stage-out file patterns / URL

MPI / app args

Walltime estimate
Measured run time
nodes
ranks / node
hyperthreads / rank
hyperthreads / core
environment variables
command-line arguments

Executable Info

application name
(points to registered executable)

preprocess script
postprocess script
post: error handle flag
post: timeout handle flag
auto timeout retry count

BalsamJob Fields

ID / provenance

Unique ID
Name
Workflow name
Description
State history

Data flow

Parent Jobs
Working directory
Input file patterns
Stage-in file patterns / URL
Stage-out file patterns / URL

MPI / app args

Walltime estimate
Measured run time
nodes
ranks / node
hyperthreads / rank
hyperthreads / core
environment variables
command-line arguments

Executable Info

application name
(points to registered executable)

preprocess script
postprocess script
post: error handle flag
post: timeout handle flag
auto timeout retry count

Hello World

```python
from mpi4py import MPI
comm = MPI.COMM_WORLD

rank = comm.Get_rank()

print("Hello from", rank)
raise RuntimeError
```

hello.py

Hello World

```python
from mpi4py import MPI
comm = MPI.COMM_WORLD

rank = comm.Get_rank()

print("Hello from", rank)
raise RuntimeError
```

hello.py

```
~$  balsam app --exec hello.py --name hello --desc 'say hello and fail'
```

Hello World

```python
from mpi4py import MPI
comm = MPI.COMM_WORLD

rank = comm.Get_rank()

print("Hello from", rank)
raise RuntimeError
```

hello.py

```
~$  balsam app --exec hello.py --name hello --desc 'say hello and fail'

Application:
------------
PK:             5
Name:           hello
Description:    say hello and fail
Executable:     /Users/misha/anaconda3/envs/testmpi/bin/python /Users/misha/hello.py
Preprocess:
Postprocess:
Envs:

Added app to database
```

Hello World

```
~$  balsam job --name test --app hello --num-nodes 1 --ranks-per 4 --workflow SDL --wall-minutes 1
```

```
Balsam Job
----------
ID:                     a503c665-ecd8-49e9-8c32-e6b251739531
name:                   test
workflow:               SDL
latest state:           [03-01-2018 13:34:25.951365 CREATED]
description:
```

Hello World

```
~$  balsam job --name test --app hello --num-nodes 1 --ranks-per 4 --workflow SDL --wall-minutes 1
```

```
Balsam Job
----------
ID:                     a503c665-ecd8-49e9-8c32-e6b251739531
name:                   test
workflow:               SDL
latest state:           [03-01-2018 13:34:25.951365 CREATED]
description:
```

```
~$  balsam ls
```

```
                                job_id | name | workflow | application |   state
-------------------------------------------------------------------------------
a503c665-ecd8-49e9-8c32-e6b251739531 | test | SDL      | hello       | CREATED
```

```
$  balsam launcher --consume
```

Hello World

```
$  balsam ls --hist
```

```
[03-01-2018 13:39:38.084306 CREATED]
[03-01-2018 13:39:45.964650 READY] dependencies satisfied
[03-01-2018 13:39:45.987540 STAGED_IN]
[03-01-2018 13:39:46.991980 PREPROCESSED] No preprocess: skipped
[03-01-2018 13:39:47.993910 RUNNING]
[03-01-2018 13:39:49.991323 RUN_ERROR] MPIRunner [test | 0801e7d9] RETURN CODE 1:
    that caused that situation.
    ----------------------------------------------------------------------
    real 1.23
    user 0.33
    sys 0.15

[03-01-2018 13:39:50.011225 FAILED] No error handler: run failed
```

Hello World

```
Hello from 1
Hello from 2
Hello from 3
Hello from 0
Traceback (most recent call last):
  File "/Users/misha/hello.py", line 7, in <module>
    raise RuntimeError
```

test.out

```
01-Mar-2018 13:39:47|19067|   DEBUG|balsam.launcher.runners:346] 0 single-process jobs can run
01-Mar-2018 13:39:47|19067|   DEBUG|balsam.launcher.runners:353] 1 MPI jobs can run; largest requires 1 nodes
01-Mar-2018 13:39:47|19067|    INFO|balsam.launcher.runners:385] Running 1-node MPI job
01-Mar-2018 13:39:47|19067|    INFO|balsam.launcher.runners:165] MPIRunner [test | 0801e7d9] Popen:
time -p ( mpirun -n 4 -npernode 4 -x CONDA_PYTHON_EXE="/Users/misha/anaconda3/bin/python" -x DJANGO_SETTINGS_MODULE="balsam.django_conf
ig.settings" -x BALSAM_DB_PATH="/Users/misha/workflow/argobalsam/default_balsamdb" -x BALSAM_JOB_ID="0801e7d9-9935-4554-8770-3f6f4678e2
3c" -x BALSAM_PARENT_IDS="[]" -x BALSAM_CHILD_IDS="[]"   /Users/misha/anaconda3/envs/testmpi/bin/python /Users/misha/hello.py )
01-Mar-2018 13:39:47|19067|    INFO|balsam.launcher.runners:166] MPIRunner: writing output to /Users/misha/workflow/argobalsam/default_
balsamdb/data/SDL/test_0801e7d9/test.out
01-Mar-2018 13:39:47|19067|   DEBUG|balsam.launcher.runners:391] Using workers: [0]
```

balsam.log

Hyperparameter Optimization

https://scikit-optimize.github.io/notebooks/ask-and-tell.html

```
for i in range(10):
    next_x = opt.ask()
    f_val = objective(next_x)
    opt.tell(next_x, f_val)

plot_optimizer(opt, x, fx)
```

Async Bayesian optimization loop

Expensive hyperparameter evaluation loosely coupled to optimizer via Balsam

```
import balsam.launcher.dag as dag
from balsam.service.models import BalsamJob, END_STATES
```

```python
def create_job(x, eval_counter, cfg):
    '''Add a new benchmark evaluation job to the Balsam DB'''

    jobname = f"task{eval_counter}"
    cmd = f"{sys.executable} {cfg.benchmark_filename}"
    args = ' '.join(f"--{p}={v}"
                    for p,v in zip(cfg.params, x)
                    if 'hidden' not in p
                   )
    envs = f"KERAS_BACKEND={cfg.backend}"

    child = dag.spawn_child(
                name = jobname,
                direct_command = cmd,
                application_args = args,
                environ_vars = envs,
                wall_time_minutes = 2,
                num_nodes = 1, ranks_per_node = 1,
                wait_for_parents = False
              )
    print(f"Added task {eval_counter} to job DB")
    print(cmd, args)
    return child.job_id
```

```
# Read in new results
new_jobs = BalsamJob.objects.filter(job_id__in=my_jobs.keys())
new_jobs = new_jobs.filter(state="JOB_FINISHED")
new_jobs = new_jobs.exclude(job_id__in=finished_jobs)
for job in new_jobs:
    try:
        result = read_result(job, my_jobs)
    except FileNotFoundError:
        print(f"ERROR: could not read output from {job.cute_id}")
    else:
        resultsList.append(result)
        print(f"Got data from {job.cute_id}")
        pprint(result)
        x, y = result['x'], result['cost']
        opt.tell(x, y)
```

Hyperparameter Optimization

- Model training jobs are loosely coupled to hyperparameter optimization
- A wide variety of models can be optimized under a single framework
- Optimization is resilient to job timeouts, unexpected errors, and is trivially checkpointable

Summary

- https://xgitlab.cels.anl.gov/datascience/balsam

- New features in Balsam make it a useful tool for managing ensemble workflows

- Workflow tasks can be loosely coupled and run asynchronously across multiple batch jobs

Argonne NATIONAL LABORATORY