Using Balsam for Ensemble Workflows

Misha Salim
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 Components

Login

Mom

Compute

balsam job

balsam ls

Balsam DB
Balsam Components

- Balsam Components
- Balsam DB
- Login
- Mom
- Compute
- Metascheduler
  - Query BalsamJob DB
  - Query Cobalt queues
  - Package BalsamJobs for concurrent execution

balsam service
Balsam Components

Metascheduler

Login

Cobalt

Balsam DB

Launcher

Pre-processing
Data flow
Execute MPI jobs
MPI-package serial jobs
Timing/instrumentation
Post-processing
Error/timeout handlers

Mom

Compute
Balsam Components

Balsam Components diagram:
- Metascheduler
- Cobalt
- Launcher
- Balsam DB
- Compute
- Login
- aprun
- Job 1
- Job 2
- Job N
Balsam Components

Balsam Components Diagram:

- Login
- Balsam DB
- Launcher
- NO SERVICE
- aprun
- aprun
- aprun
- aprun
- aprun
- aprun
- aprun
- aprun
- aprun
- aprun
- aprun

Jobs:
- Job 1
- Job 2
- Job N
- ...
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

Executable Info
- application name
  (points to registered executable)
- preprocess script
- postprocess script
- post: error handle flag
- post: timeout handle flag
- auto timeout retry count

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

**Executable Info**
- application name
  (points to registered executable)
- preprocess script
- postprocess script
- post: error handle flag
- post: timeout handle flag
- auto timeout retry count

**MPI / app args**
- Walltime estimate
- Measured run time
- # nodes
- ranks / node
- hyperthreads / rank
- hyperthreads / core
- environment variables
- command-line arguments
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

**Executable Info**
- application name (points to registered executable)
- preprocess script
- postprocess script
- post: error handle flag
- post: timeout handle flag
- auto timeout retry count

**MPI / app args**
- Walltime estimate
- Measured run time
- # nodes
- ranks / node
- hyperthreads / rank
- hyperthreads / core
- environment variables
- command-line arguments
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

**Executable Info**
- application name
  (points to registered executable)
- preprocess script
- postprocess script
- post: error handle flag
- post: timeout handle flag
- auto timeout retry count

**MPI / app args**
- Walltime estimate
- Measured run time
- # nodes
- ranks / node
- hyperthreads / rank
- hyperthreads / core
- environment variables
- command-line arguments
from mpi4py import MPI
comm = MPI.COMM_WORLD

rank = comm.Get_rank()

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

```python
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print("Hello from", rank)
raise RuntimeError
```

```bash
$ 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
```

```
~$ 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

<table>
<thead>
<tr>
<th>Balsam Job</th>
<th>a503c665-ecd8-49e9-8c32-e6b251739531</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID:</td>
<td>test</td>
</tr>
<tr>
<td>name:</td>
<td>test</td>
</tr>
<tr>
<td>workflow:</td>
<td>SDL</td>
</tr>
<tr>
<td>latest state:</td>
<td>[03-01-2018 13:34:25.951365 CREATED]</td>
</tr>
<tr>
<td>description:</td>
<td></td>
</tr>
</tbody>
</table>
```
Hello World

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

<table>
<thead>
<tr>
<th>job_id</th>
<th>name</th>
<th>workflow</th>
<th>application</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>a503c665-ecd8-49e9-8c32-e6b251739531</td>
<td>test</td>
<td>SDL</td>
<td>hello</td>
<td>CREATED</td>
</tr>
</tbody>
</table>

$ 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 13:39:47.993910  RUNNING]
    that caused that situation.

-----------------------------------------------------------------------------------
real 1.23
user 0.33
sys 0.15

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

balsam.log
Hyperparameter Optimization

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

```python
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

```python
import balsam.launcher.dag as dag
from balsam.service.models import BalsamJob, END_STATES
```
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\}"علوم
        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