Real-time HEP analysis with funcX:
A high-performance platform for function as a service

Yadu Babuji, Ben Blaiszik, Kyle Chard, Ryan Chard, Ian Foster, Daniel Katz, Zhuozhao Li, Tyler Skluzacek, Ana Trisovic, Anna Woodard

05 November 2019
CHEP 2019
What is funcX?

FuncX is an open-source platform which allows users to register, discover, and execute functions on arbitrary computing endpoints.

Users interact with funcX via a REST API exposed by an AWS-hosted funcX service.
Note: What is described and evaluated in this talk is a prototype and under active development—stay tuned! If you’d like to try it out, contact me to be whitelisted.
What is a funcX endpoint?

**FuncX endpoint**: abstraction of a computational resource (a local machine, cluster, cloud, or supercomputer). The endpoint agent allows the funcX service to dispatch functions to that resource.

Admins or users can register and deploy an endpoint for themselves and/or others.

Scaling strategy can be customized to minimize latency (always keep a specified number of nodes provisioned) or to maximize efficiency (provision nodes in proportion to pending tasks).
A simple example: start an endpoint

1. Install funcX
2. Configure the endpoint (default configuration will run functions locally, or specify a Condor/Slurm/Torque/etc cluster, or specify AWS/Azure/GoogleCloud)
3. Authenticate and start endpoint

~ >pip install funcx==0.0.1a2
~ >funcx-endpoint configure my_endpoint
A default profile has been create for <my_endpoint> at /afs/crc.nd.edu/user/awoodard/.funcx/my_endpoint/config.py
Configure this file and try restarting with:
  > funcx-endpoint start my_endpoint
~ >funcx-endpoint start my_endpoint
It looks like this is the first time you're accessing this service.
Please log in to Globus at this link:
https://auth.globus.org/v2/oauth2/authorize?client_id=4cf29807-cf21-49ec-9443-ff9a3fb9f81c&redirect_uri=[...]
Copy and paste the authorization code here: XXXXXXXXXXXXXXXXXXX
Thanks! You're now logged in.
from funcx.sdk.client import FuncXClient

client = FuncXClient()

def compute_sum(items):
    return sum(items)

func_uuid = client.register_function(
    compute_sum,
    description="A sum function"
)

payload = [1, 2, 3, 4, 66]
endpoint_uuid = 'f8696260-c060-4f2b-814f-f5ba917f8472'
task_id = client.run(
    payload,
    endpoint_id=endpoint_uuid,
    function_id=func_uuid
)

result = client.get_result(task_id)  # result is now 76
What happens when you execute a function?

FuncX main components:

1. a registry of endpoints
2. a registry of functions (and optionally, associated containers)
3. a cloud-hosted system for management of function execution.
How does the funcX prototype scale?

**Strong scaling:** total concurrent functions fixed to 100k.

**Weak scaling:** functions per container fixed to 10.

Scales to 130k+ containers; good performance up to \(\sim 2k\) containers (1 second function) or \(\sim 16k\) containers (1 minute function); similar performance using Singularity (on Theta) and Shifter (on Cori).
How can funcX speed up time-to-insight for physicists?

1. Registry of functions+containers\(^1\) reduces opportunities for users to make mistakes
   Added bonuses: improve reproducibility, encourage modularity!

2. Combine and utilize resources where they are available—backfill queues, non-dedicated campus clusters, etc

3. Scale interactive analysis in Jupyter notebooks

4. Use appropriate hardware where needed
   For example: dispatch machine learning tasks to GPUs.

5. Simple python SDK instead of writing submit scripts
   Code to be executed is factorized from details of execution environment— to run somewhere else, simply swap out the endpoint UUID.

\(^1\)Container support is currently being refactored in the prototype.
Show me the physics!
Coffea\(^2\) uses columnar operations to provide 1) an array-based syntax for manipulating HEP event data— implements histogramming, plotting, transformations, corrections, etc; and 2) a unified interface for writing executors which facilitate horizontal scaling.

To demonstrate how funcX can be used for real physics analyses, we wrote a funcX executor for Coffea\(^3\).

\(^2\)Columnar Object Framework For Effective Analysis– check out their talk! https://indico.cern.ch/event/773049/contributions/3476048/

\(^3\)Currently lives in a forked repo– will be merged after finalization: https://github.com/annawoodard/coffea
The funcX coffea backend:

- Registers a function which takes input data and runs an analysis processor over it
- Transfers the analysis processor to each worker once and caches it
- Stages data out via XrootD (for now– more stageout methods can be added)
- Provides a convenience wrapper which chunks the data and submits a function for each chunk, then passes the results back to Coffea to combine into the final result histograms/counts
import os

from funcx.config import Config
from funcx.strategies import SimpleStrategy
from parsl.providers import CondorProvider
from parsl.executors import HighThroughputExecutor
from parsl.addresses import address_by_hostname

proxy = '/tmp/x509up_u{}' .format(os.getuid())

worker_init = ""
source /cvmfs/sft.cern.ch/lcg/views/LCG_95apython3/x86_64-centos7-gcc7-opt/setup.sh
export PATH=~/.local/bin:$PATH
export PYTHONPATH=~/.local/lib/python3.6/site-packages:$PYTHONPATH
export X509_USER_PROXY=`pwd`/{}
""".format(os.environ['USER'], os.path.basename(proxy))

config = Config(
    scaling_enabled=True,
    cores_per_worker=1,
    provider=CondorProvider(
        cores_per_slot=8,
        init_blocks=50,
        max_blocks=50,
        worker_init=worker_init,
        transfer_input_files=[proxy]
    ),
)

Step 1: start endpoints at Notre Dame and Wisconsin.
import json

from coffea.processor import run_funcx_job
from coffea.processor.funcx.executor import funcx_executor

import funcx
funcx.set_file_logger('/afs/crc.nd.edu/user/a/awoodard/funcx.log')

ndt3_uuid = '81404f4b-9b35-4b92-9881-a02fe5e52693'
wisconsin_uuid = 'af21d0db-27f2-4906-beba-6baffac18393'
chunksize=750000

with open('metadata/samplefiles.json') as f:
    datasets = json.load(f)['Hbb_2017']

treenames = ['otree', 'Events']  # process mixed skims and full trees
stageout_path = 'root://deepthought.crc.nd.edu://store/user/awoodard/funcx'

final_accumulator = run_funcx_job(
    [ndt3_uuid, wisconsin_uuid],  # Add as many endpoints as you like!
    datasets,
    treenames,
    'boostedHbbProcessor.coffea',
    funcx_executor,
    stageout_path,
    executor_args=executor_args,
    chunksize=chunksize
)

Step 2: find a real physics analysis to run—we borrowed from the coffeaandbacon H→bb analysis⁴.

Step 3: pass their analysis processor (defines analysis selections, weights, and histograms) and datasets to the run_funcx_job wrapper⁵.

⁴https://github.com/nsmith-/coffeandbacon
⁵For full Jupyter notebooks, see

https://github.com/annawoodard/coffeandbacon/blob/master/analysis/baconbits-funcx.ipynb
https://github.com/annawoodard/coffeandbacon/blob/master/analysis/baconbits-plot.ipynb
Case study: real-time HEP analysis with Coffea and funcX

Result: processed \(~291\) million events (nanoAOD format) in 9 minutes \((1.9 \mu s/\text{event})\) on \(~400\) cores\(^4\), combining resources from two separate sites.

Analysis code from
https://github.com/nsmith-/
coffeandbacon

\(^4\text{Compare with 7.6 minutes (1.6 }\mu \text{s/event) on }\sim400\text{ cores with Parsl}\)
Conclusions

• FuncX is an open-source platform which allows users to decompose applications into collections of functions that can each be executed in the best location (in terms of cost/execution time/resource availability), on endpoints managed by users or admins

• FuncX’s registry of functions+containers can improve modularity and reproducibility in user code

• We’ve implemented a funcX processing backend for the Coffea analysis framework and demonstrated good performance while integrating computing resources from multiple sites
Keep in touch!

annawoodard@uchicago.edu

https://funcx.readthedocs.io
(UNDER CONSTRUCTION)

Special thanks to Kevin Lannon, Kenyi Hurtado, Paul Brennar, and others at Notre Dame, and Chad Seys and others at Wisconsin, for help with site testing; and to Lindsey Gray and the Coffea Team for building an awesome analysis framework and providing the real-world analysis used for testing the funcX backend.