funcX

Fire-and-Forget Federated Function as a Service

Rachana Ananthakrishnan[*⚑], Yadu Babuji*[*⚑], Josh Bryan*[*⚑], Ben Clifford*[*], Kyle Chard*[*⚑], Ryan Chard*[*⚑], Ian Foster*[*⚑], Ben Galewsky*[°], Chris Janidlo*[*⚑], Kevin Hunter Kesling*[*⚑], Daniel S. Katz*[°], Zhuozhao Li*[*], Stephen Rosen*[*⚑] Tyler Skluzacek*[*], Lei Wang*[*⚑]*

[*]University of Chicago & Argonne National Laboratory; [⚑]Globus;

[°]National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign

Federated Function as a Service

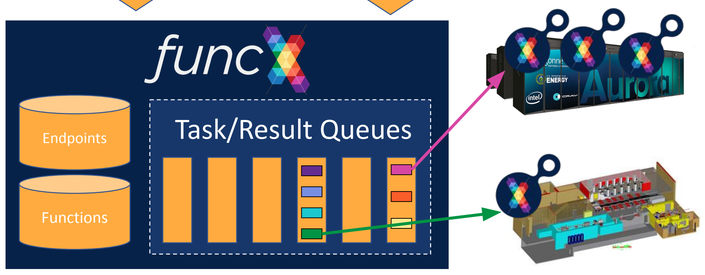
- Modern computing environments are distributed and heterogeneous; modern workloads require specialized hardware, rapid responses, and remote processing (e.g., near data)
- FaaS provides an intuitive paradigm: users register and invoke programming functions without regard for underlying infrastructure
- Federated FaaS enables functions to be dispatched to remote endpoints chosen for data locality, security, or other concerns
- funcX allows users to execute Python functions (which may invoke executables, MPI programs, etc.) on arbitrary resources (e.g., CPUs, GPUs) from short to long run times

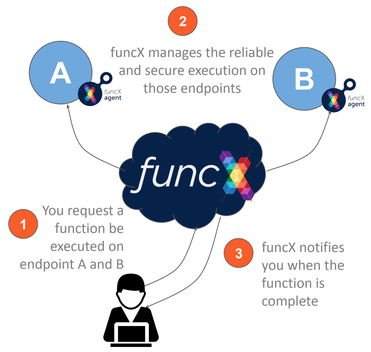
(1) Registration (function + container)

```
def compute(*args):
  # do something
  return results
```

(2) Execution (function, endpoint, args)

```
F(ep1, 6)
```





Fire-and-forget execution

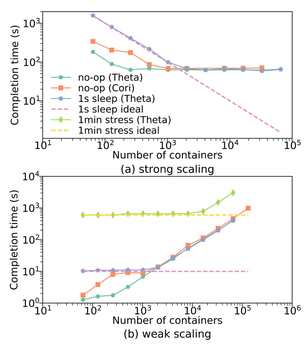
Outsource the challenging aspects of remote execution

funcX manages authentication, serialization of functions and data, reliable execution (optionally in containers), and delivery of results back to users

Transform resources into FaaS endpoints

Easily manage execution across distributed resources

The funcX endpoint software can be deployed on laptops, clouds, clusters, and supercomputers. It provisions resources elastically based on workload.



High performance

Launch millions of tasks

funcX supports batch submission and monitoring, asynchronous callbacks, container warming, automated resource scaling, fault tolerance, and prefetching

Install and configure a funcX endpoint

```
$ pip install funcx_endpoint

$ funcx-endpoint configure <ENDPOINT_NAME>

$ funcx-endpoint start <ENDPOINT_NAME>
```

Instantiate a funcX client

```
from funcx.sdk.client import FuncXClient

fxc = FuncXClient()
```

Register Python function with input args

```
def hello_world(name):
    return "Hello %s!" % name

func_uuid = fxc.register_function(hello_world)
```

Execute by specifying endpoint & input args

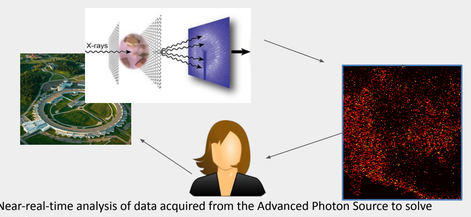
```
ep_id = '4b116d3c-1703-4f8f-9f6f-39921e5864df'
result = fxc.run(name='world',
        endpoint_id=ep_id, function_id=func_uuid)
```

Retrieve results (& exceptions)

```
fxc.get_result(result)
```

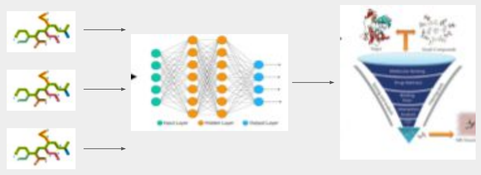
Application examples

Real-time crystallography



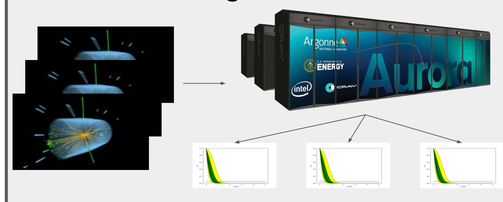
Near-real-time analysis of data acquired from the Advanced Photon Source to solve protein structures at room temperature.
(Joachimiak et al., https://doi.org/10.1073/pnas.2100170118)

ML-based drug screening



Screening billions of molecules to identify potential COVID-19 therapeutics. Computing molecule features, running ML inference, selecting top results.
(National Virtual Biotechnology Laboratory, arXiv:2006.02431)

HEP fitting as a service



Wrapping a C-based statistical inference tool as a function so scientists can easily fit multiple different hypotheses for new physics signatures (signals).
(Feickert et al., arXiv:2103.02182)

funcX Adoption (July 2022)

310,000 registered functions	**19 million** function invocations	
396 users	**3900** registered endpoints	**121s** average function runtime

funcX development has been supported by NSF 2004894/2004932 and an Argonne Lab Directed Research and Development award