Parsl & funcX: Build and test challenges

Daniel S. Katz (d.katz@ieee.org, @danielskatz)
Chief Scientist, NCSA
Associate Research Professor, CS, ECE, iSchool
University of Illinois at Urbana Champaign

SIAM PP22: Panel PD3: Build, Integration and Testing for Sustainable Scientific Computing Software; 25 February 2022

Co-authors: Yadu Babuji, Josh Bryan, Kyle Chard, Ryan Chard, Ben Clifford, Ian Foster, Ben Galewsky, Zhuozhao Li, Kirill Nagaitsev, Stephen Rosen, Mike Wilde

THE UNIVERSITY OF CHICAGO
Argonne NATIONAL LABORATORY
ILLINOIS
NCSA | National Center for Supercomputing Applications

https://doi.org/10.5281/zenodo.6210785

Parsl: A Python parallel scripting library

https://parsl-project.org

https://github.com/Parsl/parsl

Apps define opportunities for parallelism

- Python apps call Python functions
- Bash apps call external applications
- Implemented using decorators

Apps can be run remotely

Apps are asynchronous - return “futures”, a proxy for a result that might not yet be available

Apps run concurrently respecting dataflow dependencies. Natural parallel programming!

Parsl scripts are independent of where they run. Write once run anywhere!

```
pip install parsl
```

```
@python_app
def hello ():
    return 'Hello World!'

print(hello().result())
```

```
Hello World!
```

```
@bash_app
def echo_hello(stdout='echo-hello.stdout'):
    return 'echo "Hello World!"'

echo_hello().result()

with open('echo-hello.stdout', 'r') as f:
     print(f.read())
```

```
Hello World!
```

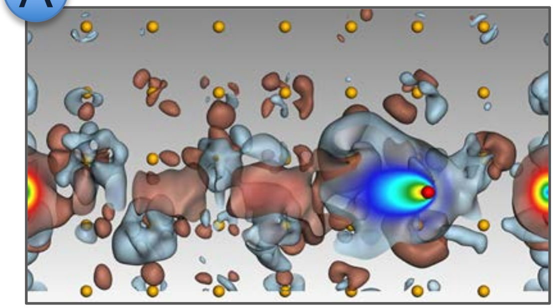
Try parsl via binder: https://parsl-project.org/binder

https://doi.org/10.5281/zenodo.6210785

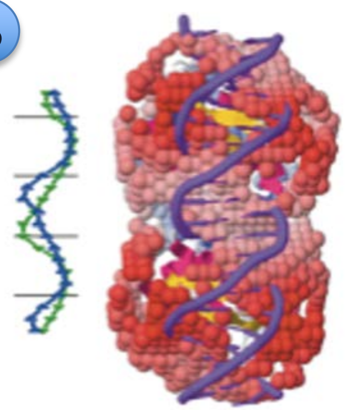
Parsl is being used in a wide range of scientific applications

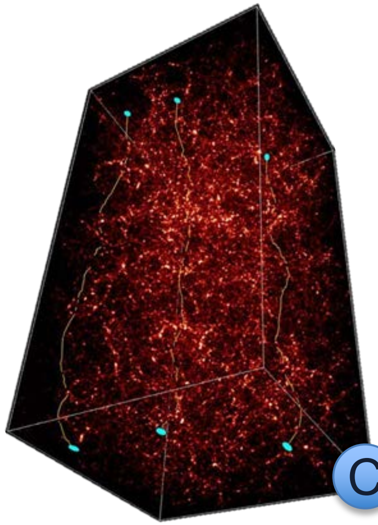
- **A** Machine learning to predict stopping power in materials
- **B** Protein and biomolecule structure and interaction
- **C** LSST simulation and weak lensing using sky surveys
- **D** Cosmic ray showers in QuarkNet
- **E** Information extraction to classify image types in papers
- **F** Materials science at the Advanced Photon Source
- **G** Machine learning and data analytics in materials

A



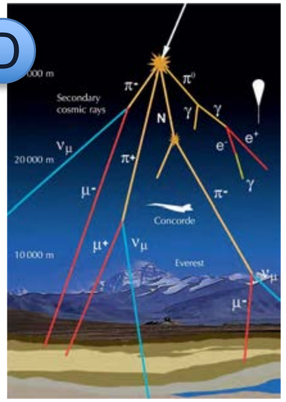
B



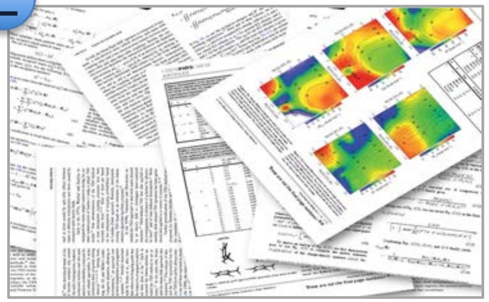


C

D

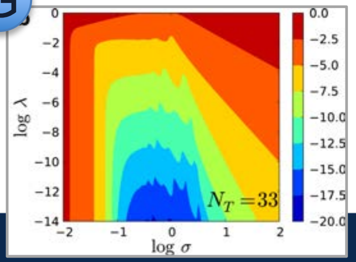


E



F

G



https://doi.org/10.5281/zenodo.6210785

funcX: Portable serverless computing for science

https://funcx.org

Turn **any** machine into a function serving endpoint

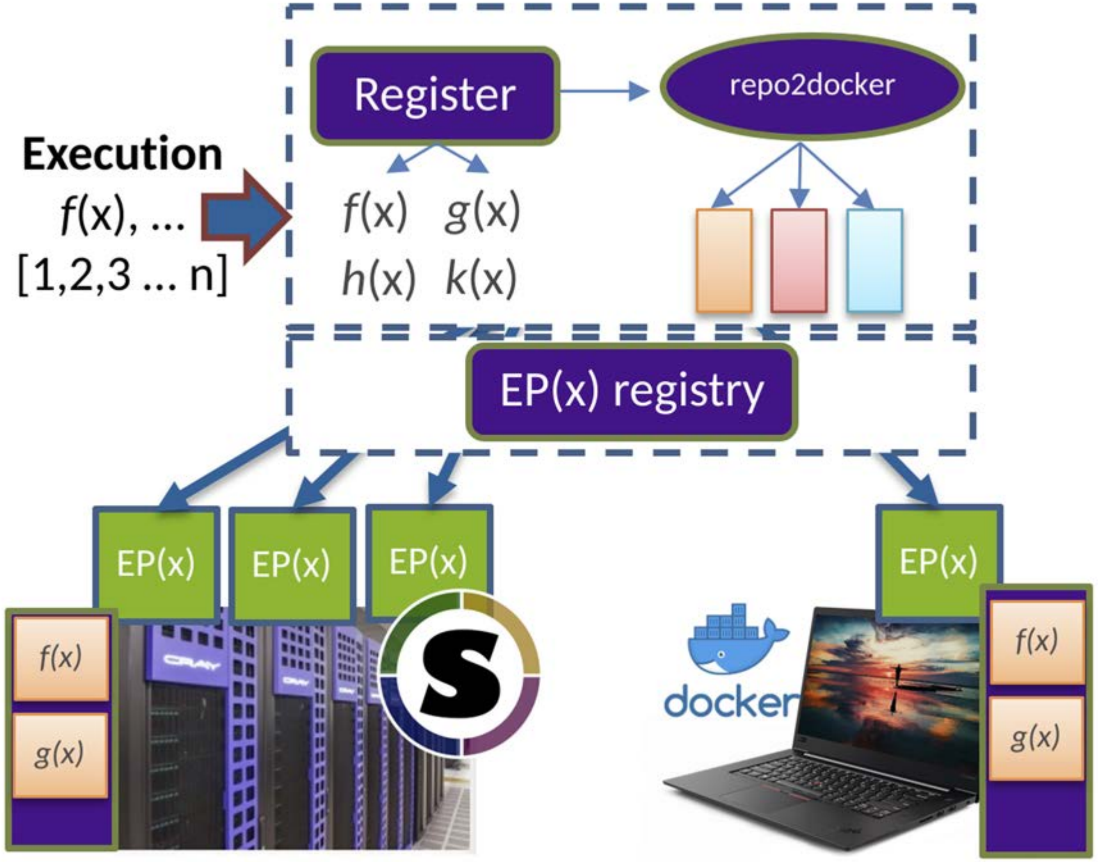
Remove barriers to using diverse and distributed infrastructure

Functions:

- Register once, run anywhere
- Can associate a container for encapsulation
- Authn/z (via Globus Auth) for user execution
- Add Globus group to a function to share it

Endpoints:

- Lightweight agent that can be deployed by users
- Abstracts underlying resource and elastically scales to demand



https://doi.org/10.5281/zenodo.6210785

funcX use cases

Metadata extraction (Xtract)

- Metadata extractors implemented as funcX functions; either distribute metadata extraction tasks to data or bring data to the cloud & act on it

Machine Learning inference (DLHub)

- Performs on-demand machine learning inference tasks, with inference requests routed to a funcX function executed within a model's container, using funcX endpoints on Kubernetes clusters that dynamically scale containers to serve ML models

Synchrotron Serial Crystallography & X-ray Photon Correlation Spectroscopy

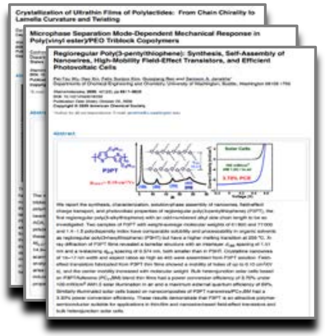
- High throughput analysis: as data are collected at the beamline, they are moved to ALCF and funcX functions are triggered to perform analyses

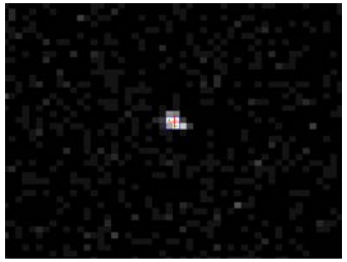
Quantitative Neurocartography

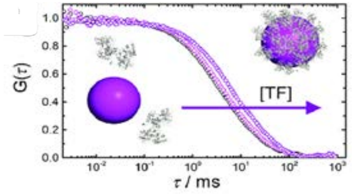
- Users invoke tasks on supercomputers via Automo library via funcX functions to create previews or perform full reconstructions w/ GB-TB of data

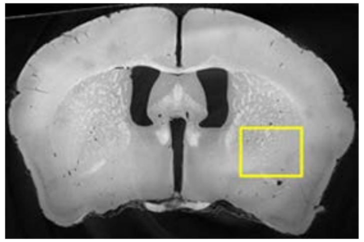
Real-time data analysis in High Energy Physics

- Aggregates histograms of analysis products in real time via Coffea framework using funcX backend to process 300m events in 9 min across 2 resources









Enabling portability in Parsl & funcX: *providers*

The same Parsl app or funcX function can be run locally, on grids, clouds, or supercomputers

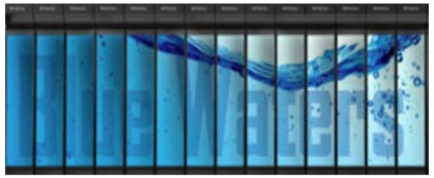
Config object specify how to run on resources; can be built by users, and are provided for common resources

- Includes: authentication method, scheduler choice, queue/job parameters, file transfer method

Growing support for various schedulers and cloud vendors

Open Science Grid

amazon web services



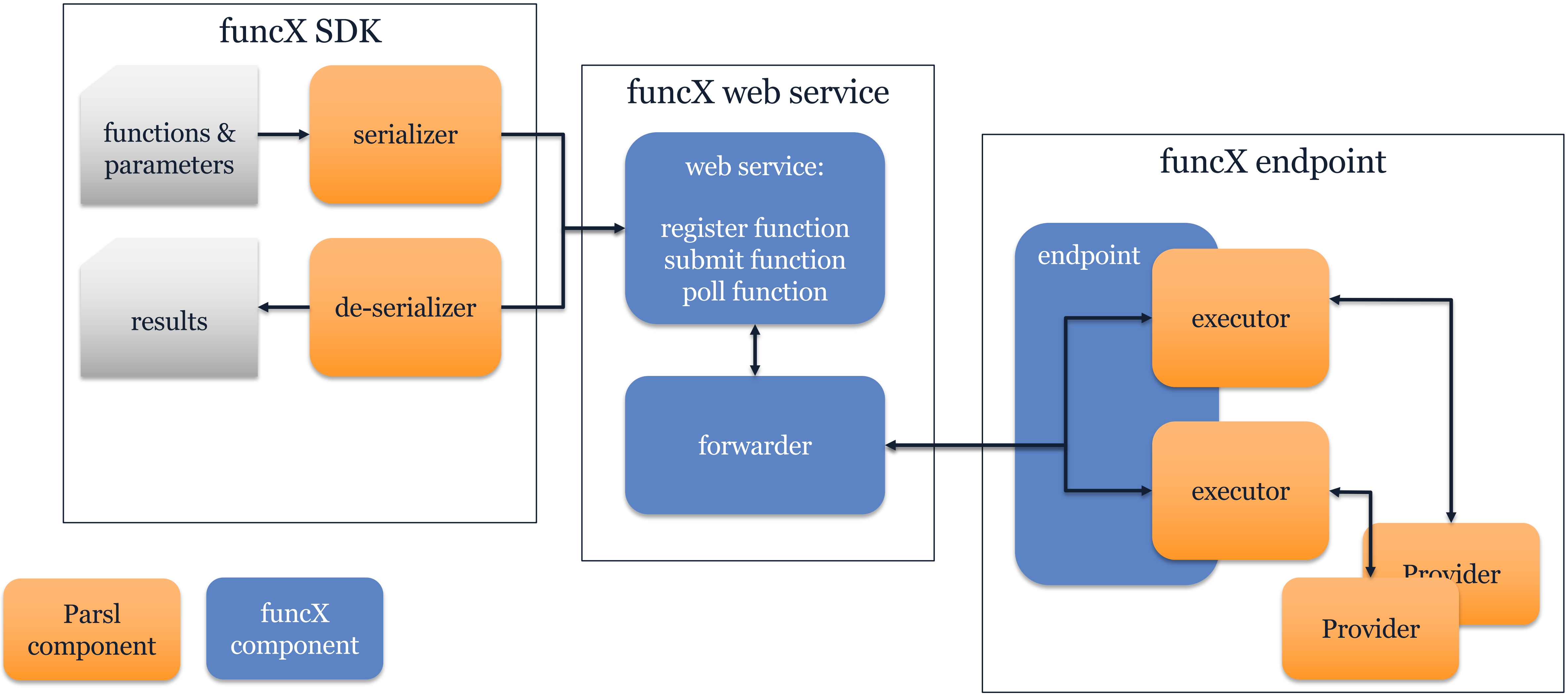
Google Cloud

XSEDE

⊟ Configuration

- How-to Configure
- Comet (SDSC)
- Cori (NERSC)
- Stampede2 (TACC)
- Theta (ALCF)
- Cooley (ALCF)
- Swan (Cray)
- CC-IN2P3
- Midway (RCC, UChicago)
- Open Science Grid
- Amazon Web Services
- Ad-Hoc Clusters
- Further help

funcX and Parsl



https://doi.org/10.5281/zenodo.6210785

Challenge: testing on real systems

- Traditional CI tries to provide a nice clean isolated environment for building software
- But we need a way to test on various HPC/cloud systems, with different schedulers, access, environments, configurations, etc.
  - We need to see how the system is set up in practice, and not be cleanly isolated from it in our CI
- Administrative/political problem:
  - In CI model, an unknown person, Sue, makes a PR to code, which triggers an attempt to run and test that code
  - As developers/maintainers, we also want to do this on a set of HPC platforms
  - HPC administrators: "Any random user can provide code and you want to automatically run it on my system? No!"

Challenge: debugging user problems

- Users have problems
- Sometimes they do things we don't expect
  - Can try to address through better documentation
- For Parsl, generally hard to know exactly what they are doing
    - We don't have access to the system they are using
    - We don't have access to their logs
    - Leads to lots of interactive discussion, very resource intensive, not scalable
- Sometimes they find bugs in our software that we can then fix, which they then need to download and install
- funcX's hybrid cloud/distributed architecture allows us to see errors that users are having, and sometimes fix things behind the scenes without needing to distribute new software

https://doi.org/10.5281/zenodo.6210785

Challenge: keeping our own software in sync

- In funcX we can struggle to keep many different things in sync: Python versions, SDK version, endpoint version. Mostly because we aren't always backwards compatible.
- We also need to sync with external dependency changes
- Keeping things in sync uses a huge amount of developer time and is even less glamorous than bug-fixing
- Hierarchy of developer excitement:
  new features > bug-fixing > packaging and release management

https://doi.org/10.5281/zenodo.6210785

Challenge: changing needs

- Initially, a newly developed product must move fast to be useful to its users
  - Requirements are discovered through use
- Later, a product must be stable be useful to its users
  - Moving too fast becomes a problem
- The "correct" balance changes over the project's lifetime

- This tension between experimental vs production isn't unique to software
- Also seen in hardware testbeds that initially don't want to support production science but eventually can be "frozen" by it

https://doi.org/10.5281/zenodo.6210785

Challenge: developer ecosystem

- Parsl funded by NSF award for initial development
  - That award is ending soon
  - How do we maintain what we've built?
    - Check/merge PRs, work with users, fix bugs, support new platforms, …
- Need for funding for core team (and others)
  - Can we depend entirely on support funding from projects that use Parsl?
    - These projects need software that works and that they can rely on
  - Are there any funding agencies that will support maintenance of existing software?
    - Funding agencies generally focus on novelty, but production software is infrastructure
    - CZI EOSS is a seemingly lonely example, and focuses on software with substantial life science impact
- And incentives for others
  - We can't easily do this by ourselves - we need systemwide changes
    - E.g., hiring and promotion policies that include software work
  - Even better, incentives encourage quality and support of contribution

https://doi.org/10.5281/zenodo.6210785

Helpful new technology: containers

- Packaging/distribution
  - One person who knows all the awkwardness and puts in the effort can be in charge of making a container image
    - An excellent machine readable way of sharing install knowledge (eg encapsulated in a Dockerfile) rather than an out-of-date wiki page that someone hacked together a while ago and no longer is right
  - Then that container image can be used in many places
    - * HPC systems have different container technology and are not as portable as one might think
- Debugging
  - Instead of getting error reports on software versions of unknown age, a user can give their container image and staff can recreate their problem on their laptop using their actual software installs

https://doi.org/10.5281/zenodo.6210785

Conclusion

- Software can be complicated, but the software itself is mostly not what makes build and test difficult
- Most difficulties are related to the environment, and lack of knowledge about it
- Normal CI on HPC has technical and policies issues
- Software as a service and containerization can help
- Also room for CS research to make this easier
- Still have to balance between changing features and stable software
- Additional difficulties are related to developer ecosystem issues
  - Funding for maintenance (it's not new and shiny, but we still need to support it)
  - Non-monetary incentives for contributors (and maintainers)

https://doi.org/10.5281/zenodo.6210785