Towards Fitting as a Service with \texttt{pyhf}

Matthew Feickert
(Ben Galewsky, Lukas Heinrich, Ricardo Rocha, Sinclert Pérez, Giordon Stark)

University of Illinois at Urbana-Champaign

Future Analysis Systems and Facilities Blueprint Workshop

October 26th, 2020
Fitting as a Service with \texttt{pyhf}

- Analysis Systems pipeline already has beta infrastructure for the final stages with \texttt{pyhf + cabinetry}
  - c.f. Alex’s talk from earlier today
  - Ask if you have questions on these projects
- Want to leverage \texttt{pyhf} hardware accelerated backends at HPC sites for real analysis speedup
  - Fitting time from hours to minutes
- HTC not target, so deploy (fitting) Function as a Service (FaaS)
  - Use API to deploy fits and return JSON output

```
$ cat benchmarks/gpu/gpu_pytorch.txt
# time pyhf cls --backend pytorch HVTWZ_3500.json
{
  "CLS_exp": [
    0.8767925199210836,
    0.17265423626183831,
    0.357231245685822,
    0.6318728762727417,
    0.8799978293609
  ],
  "CLS_obs": 0.25678098274923285
}
```

Workspace that takes over an hour on ROOT fit in under 2 minutes with \texttt{pyhf} on GPU
Open fields of exploration

- Early days in exploring solutions to implementation of Fitting as a Service
- Parallel explorations of what service and user API would look like
- funcX from Globus Labs
  - High-performance FaaS platform
  - Allows users to register and then execute Python functions in “serverless supercomputing” workflow
- Knative
  - Well adopted as a Serverless/FaaS solution on Kubernetes
  - Deployment model promotes efficient resource usage and simplifies bursting
**Sinclert** has outlined possible workflow for development (here for funcX) and end user experience.

<table>
<thead>
<tr>
<th>Development</th>
<th>Building</th>
<th>Deploying</th>
<th>Governance</th>
<th>End users</th>
<th>Fit</th>
</tr>
</thead>
<tbody>
<tr>
<td>pyhf evolves over time. Code on Github released to PyPI and conda-forge. New pyhf computations that may be interesting to expose.</td>
<td><strong>FuncX</strong> encapsulation of Python functions. Images are published to a cloud registry (DockerHub?), so they can be accessed.</td>
<td><strong>Kubernetes</strong> is used to deploy the functions. High scalability plays nicely with computational expensive workflows.</td>
<td>Governance model required. Someone needs to coordinate new deployments across the stack. In addition to enable / disable access through an auth DB.</td>
<td>Ask for access: Access request to the service. Given the amount of computing power the service could use, auth is required. Some ticketing procedure must be defined (GitHub issues?).</td>
<td>Users send HTTP requests. Users query the service, with some basic auth information. Service validates user auth before proceeding forward.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GitHub</th>
<th>PyPI</th>
<th>conda-forge</th>
<th>Docker</th>
<th>Cloud Registry</th>
<th>Kubernetes</th>
<th>Auth database</th>
<th>Continuous effort</th>
<th>Continuous effort</th>
<th>Pods</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram showing development, building, deploying, and governance processes along with end user access and fit considerations.](image)
Ben has built **prototype workflow** for fitting models from pyhf pallet for **published ATLAS SUSY 1Lbb analysis**

- Currently deployed on **Chicago River HPC cluster**
- Example implementation of Sinclert’s model

- Uses Python driver with globus for authentication

- Lukas and Matthew have tested and are able to fit all models in analysis (125 signal patches) in just under 2 minutes 30 seconds

  - N.B. Wall time includes downloading pyhf pallet from HEPData, starting funcX, sending data to funcX, and fits
  - Currently CPU, but parallelization gives significant speedup

- For working prototype, this is already a win!
**Knative + GPU Workloads**

- **Ricardo** has built prototype scaling out from CERN to Google Cloud Platform (GCP)
  - Especially interesting for GPUs/TPUs
- Supports fast auto scaling of workloads (secs) and clusters (mins) to meet demand
- **Ricardo** and **Lukas** working on version that allows per second reporting instead of per script execution

---

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: autoscale-go
  namespace: default
spec:
  template:
    metadata:
      annotations:
        # Target 10 in-flight-requests per pod.
        autoscaling.knative.dev/target: "10"
    spec:
      containers:
      - image: rochaporto/fitting:cuda11.0
```

---

**Example toy fit run on all available GCP cards**

<table>
<thead>
<tr>
<th>Google Cloud Platform GPU Cards</th>
<th>Approximate fit time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>0.5</td>
</tr>
<tr>
<td>T4</td>
<td>1.0</td>
</tr>
<tr>
<td>K80</td>
<td>1.5</td>
</tr>
<tr>
<td>P100</td>
<td>2.0</td>
</tr>
<tr>
<td>V100</td>
<td>2.5</td>
</tr>
<tr>
<td>A100</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Fitting as a Service with pyhf is a natural progression of the final stage of Analysis Systems pipeline deployed to HPC sites.

- Consumes pyhf Python API
- Allows for parallelization of fitting models from pyhf pallets across HPC/GPU systems
- Possible interface with cabinetry?

Very early stage of development, but given the relative stability of pyhf API should be able to develop and explore idea space quickly.

- pyhf not yet at v1.0, but relevant API is rather stable

Working deployment on funcX, trial deployments at CERN with Knative

End user API currently under design iteration

- Small library to give service agnostic CLI API as well as Python API?
- Scope will dictate more as projects evolve
**Ricardo:** GCP vs. Azure for single and double precision

### Single precision

<table>
<thead>
<tr>
<th>GPU Cards</th>
<th>Approximate fit time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M60</td>
<td>GCP</td>
</tr>
<tr>
<td>P4</td>
<td>Azure</td>
</tr>
<tr>
<td>P40</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td></td>
</tr>
<tr>
<td>K80</td>
<td></td>
</tr>
<tr>
<td>P100</td>
<td></td>
</tr>
<tr>
<td>V100</td>
<td></td>
</tr>
<tr>
<td>A100TPU2-8</td>
<td></td>
</tr>
</tbody>
</table>

### Double precision

<table>
<thead>
<tr>
<th>GPU Cards</th>
<th>Approximate fit time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M60</td>
<td>GCP</td>
</tr>
<tr>
<td>P4</td>
<td>Azure</td>
</tr>
<tr>
<td>P40</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td></td>
</tr>
<tr>
<td>K80</td>
<td></td>
</tr>
<tr>
<td>P100</td>
<td></td>
</tr>
<tr>
<td>V100</td>
<td></td>
</tr>
<tr>
<td>A100</td>
<td></td>
</tr>
</tbody>
</table>