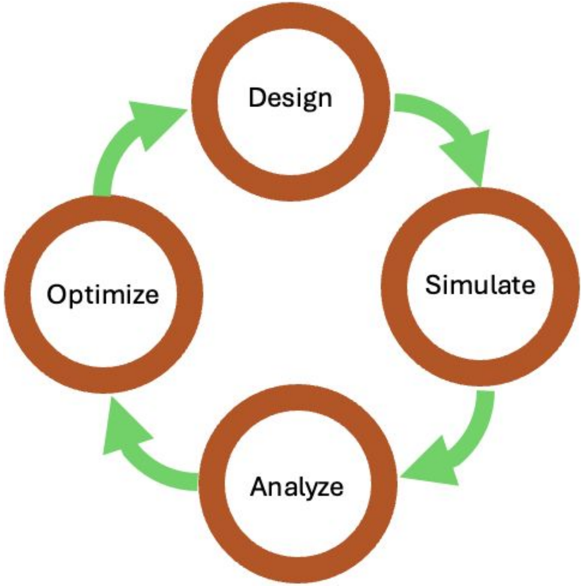
Scalable AI-assisted Workflow Management for EIC Detector Design Across Distributed Heterogeneous Resource with PanDA-iDDS

Wen Guan, Amit Bashyal, Fangying Tsai and Torre Wenaus on behalf of AID2E and PanDA teams

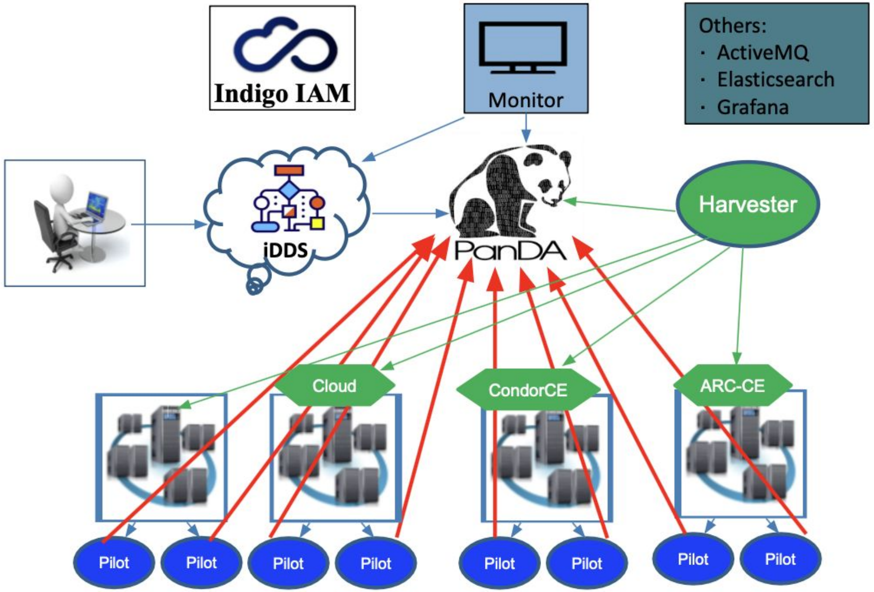
Oct, 2025

Challenge and motivation

- Detector design involves many optimization parameters and complex constraints.
- Simulations are computationally expensive.
- Need automation, parallelization, and scalability.
- AI-driven optimization of detector design demands scalable, automated workflows across distributed resources



Distributed Computing with PanDA

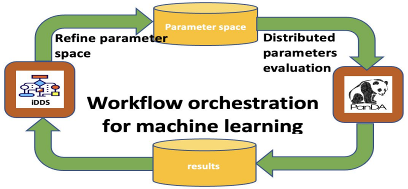


- **[PanDA (Production and Distributed Analysis system)]**
  - Unified workload management system for distributed computing
  - General interface for users, one authentication for all sites
  - Integrate different resource providers (Grid, Cloud, k8s, HPC and so on), hide the diversities from users, large scale
- **Distributed Users**
  - Users from different universities/labs can run jobs on PanDA through a http service
  - X509 or OIDC for authorization
  - LHC ATLAS: 170 sites and several thousand users
- **Distributed computing resources**
  - Diverse locations
  - Different software (slurm,condor,pbs)
  - Site differences will increase user complexity
- **Proven in LHC ATLAS and Rubin Observatory, now extended to EIC.**

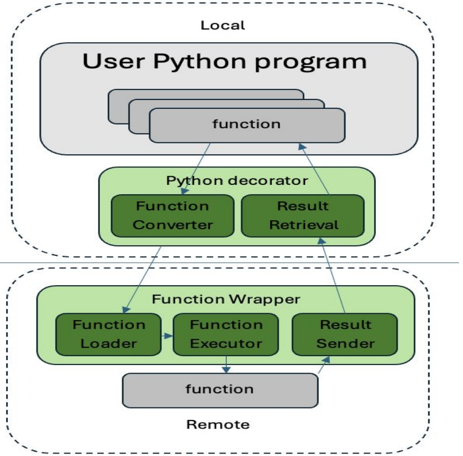
Workflow Automation with iDDS

arXiv:2510.02930

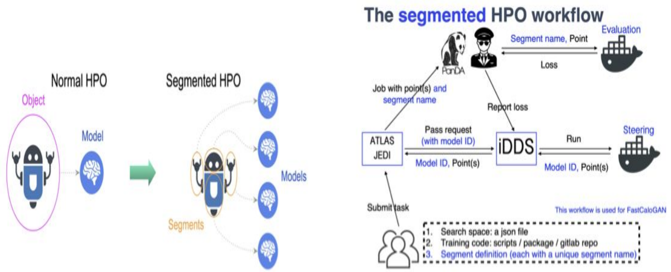
- **iDDS (intelligent Distributed Dispatch and Scheduling) orchestrates workflows for automation in PanDA**
  - Directed Acyclic Graph (DAG) management.
  - Dynamic workflows supports conditions and loops
  - Iterative workflow for machine learning integration
  - Function-as-a-Task via Python decorators
  - Asynchronous results handling
  - Integrated with PanDA for scalable computing resources



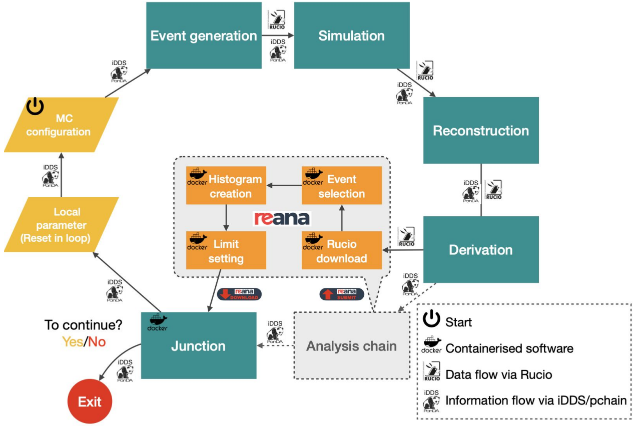
Iterative workflow



Function-as-a-Task



HyperParameter Optimization



ATLAS Active Learning

AI-assisted Detector Design for EIC (AID2E)

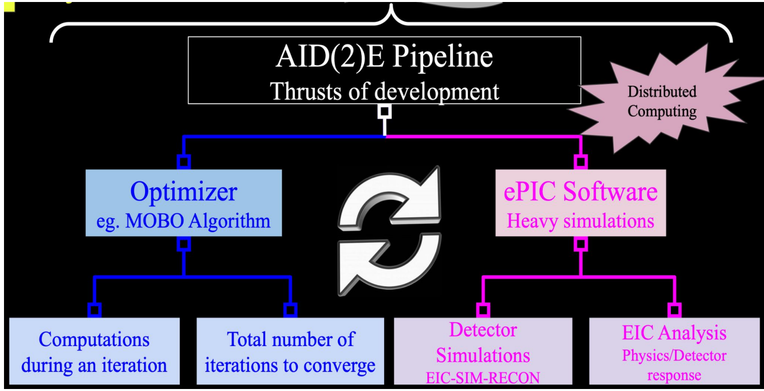
[AI-assisted detector design for the EIC (AID(2)E) 2024]

- AID2E (A scalable and distributed AI-assisted Detector Design for the EIC)
  - Contribute to advance the multiple objective optimization for detector design
    - AI-driven detector simulation, evaluation and decision optimization workflow
      - A holistic optimization
      - Optimizing two detection systems in ePIC
        - the dual-RICH (dRICH) in the central region and the B0 detector in the forward region
    - Multiple objectives
      - Momentum resolution, θ resolution, KF efficiency, Projected θ resolution @PID
  - Develop an infrastructure that optimize detector design in a scalable and distributed manner
    - Link ML (Ax platform) with HPC (slurm)
    - Link ML (Ax platform) with distributed execution (PanDA/iDDS with Grid, Cloud, HPC and so on)

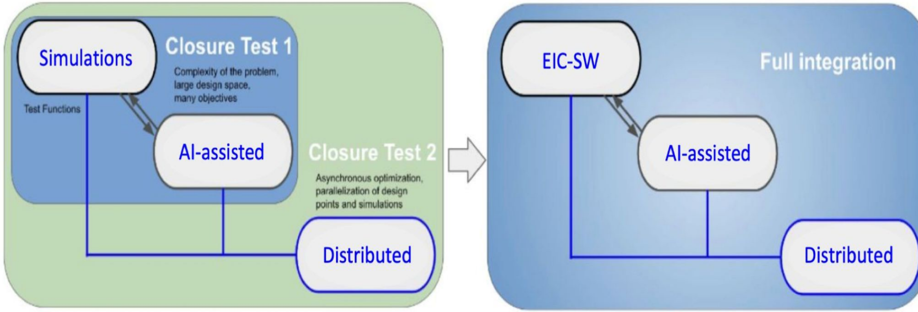
DE-FOA-0002785

AID2E Pipeline

- Multiple Objectives Optimization (MOO)
  - Multiple design parameters & objectives, complex constraints
  - Treat mapping (design parameters -> objectives) as 'black box'
  - Iterative optimization process
  - Each evaluation: Steps varying computing requirements
  - Optimization engine to minimize expensive evaluation
    - Suggest next design parameters based on history
    - Eg. bayesian optimization, genetic algorithm
- Couple MOO with ePIC software for detector simulation
- Integrate PanDA/iDDS for distributed computing

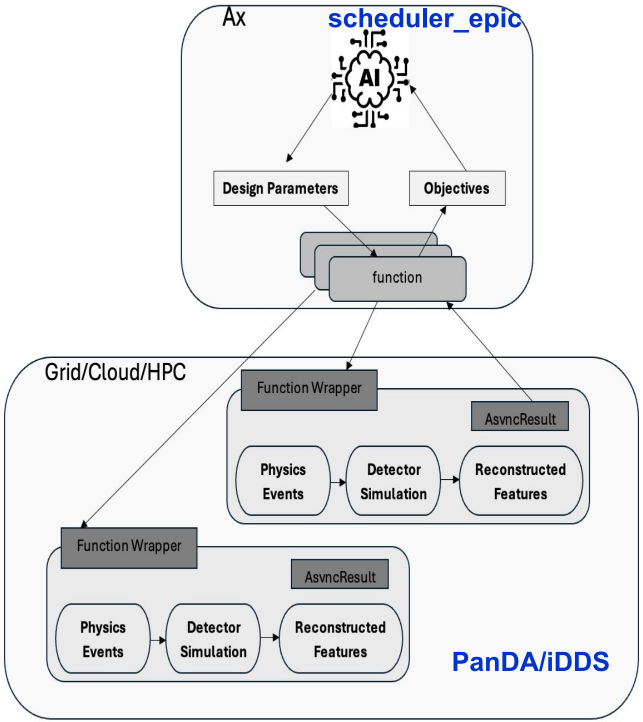


Karthik Suresh FCC 2024



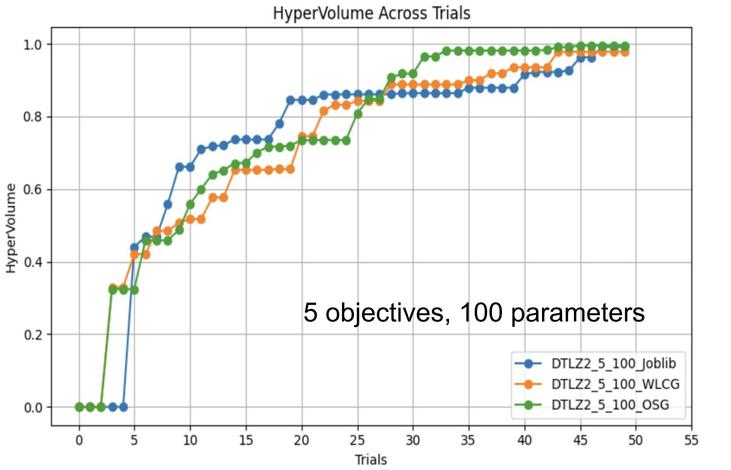
AID2E integration with PanDA/iDDS

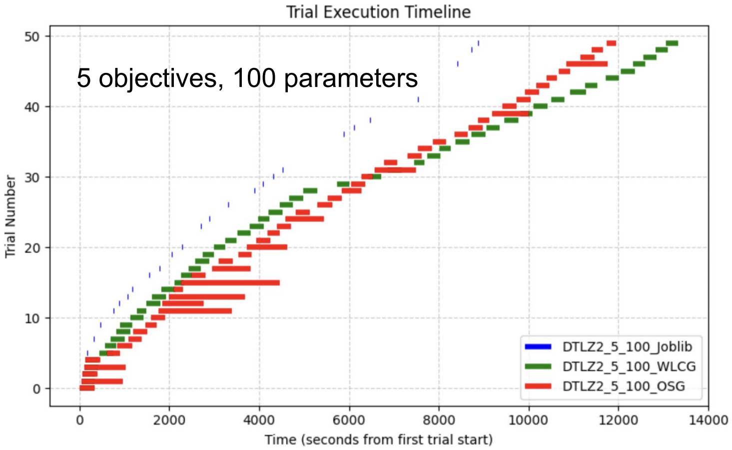
- AID2E integration with PanDA/iDDS
  - Employ PanDA/iDDS to manage AID2E parameter optimization tasks on distributed resources
  - iDDS Function-as-a-Task maps Ax objective functions to tasks and asynchronously manages the results
  - PanDA manages the tasks on distributed computing resources such as WLCG, OSG, HPC and etc
- Implementation
  - Scheduler_epic (https://aid2e.github.io/scheduler_epic/) with Ax (Adaptive Experiment Platform), manages the parameters generation and results evaluation
- Makes use of BNL PanDA instance
  - Supports multiple experiments and projects, R&D, prototyping
  - WLCG, OSG, HPC, Cloud, Kubernetes



Case study 1: DTLZ2 (A closure test)

- DTLZ2 (Deb–Thiele–Laumanns–Zitzler Problem 2): benchmark optimization problem
- Setup
  - Objectives: 3, 4, 5
  - Parameters: 20, 50, 100
- Computing resources
  - Local: Joblib (single-core)
  - PanDA distributed resources enables concurrency: WLCG BNL site and OSG
- Results analysis:
  - Similar optimization results
  - For DTLZ2, the bottleneck is to generate the trials (hyperparameters)
    - The evaluation function is fast
    - PanDA enables to run multiple concurrent trials. However, most of the time it cannot reach enough concurrent trials (some details in backup)
- Demonstrated successful integration of PanDA/iDDS for AID2E





Case study 2: dRICH-MOBO

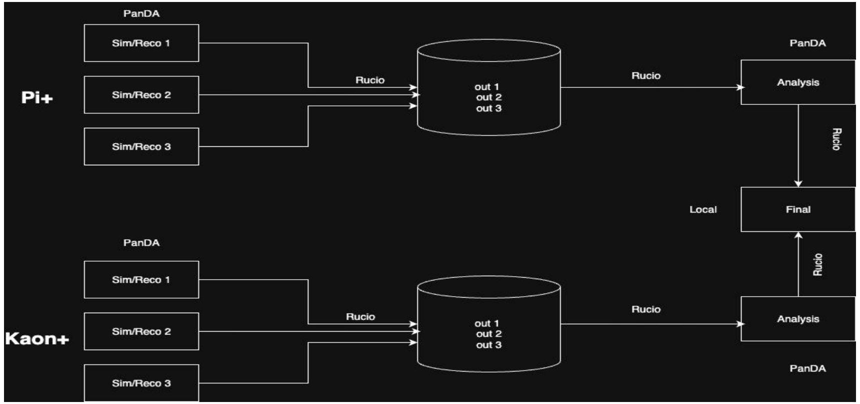
- Multiple Objectives Bayesian Optimization for optimizing the dRICH detector design
  - Multiple steps
    - Detector Simulation (PanDA Runner)
    - Objective Analysis (PanDA Runner)
    - Finalize to combine the results (Local Runner)

Chained together with PanDA workflow

Function-as-a-Task, to automatically get analysis results

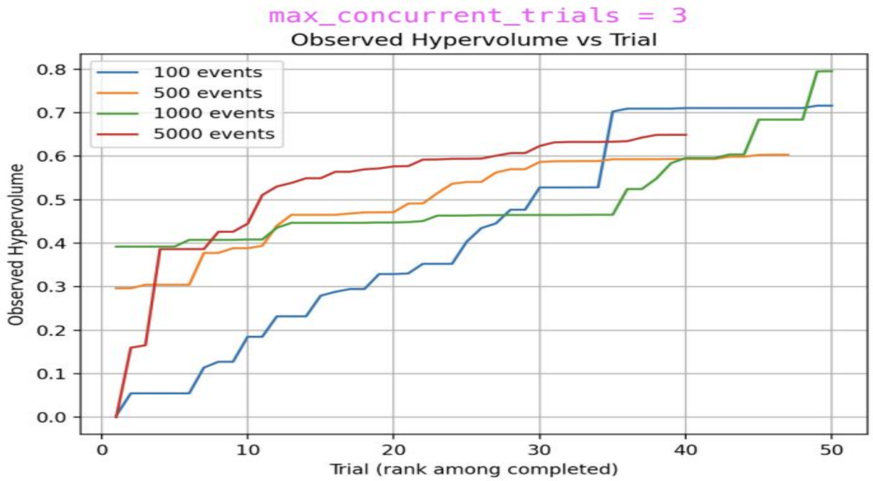
- Demonstrated successful integration of PanDA/iDDS for a complex, real detector model
- Achieved improved throughput, automation and workflow scalability across distributed resources

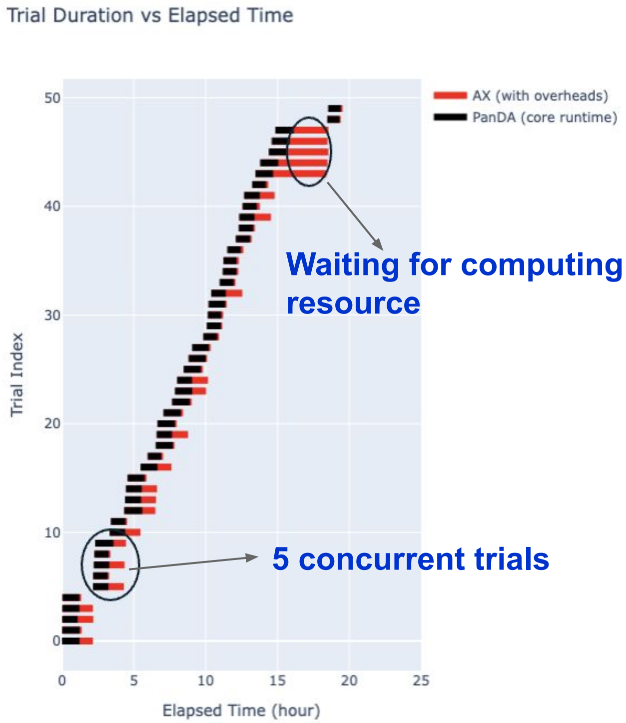
```
Trial (1 of 30)
 ├── Sim/Reco Stage
 │    ├── Pions (2 momentum bins)
 │    │    ├── Bin1 → 5 jobs of 1000 events
 │    │    └── Bin2 → 5 jobs of 1000 events
 │    └── Kaons (2 momentum bins)
 │         ├── Bin1 → 5 jobs of 1000 events
 │         └── Bin2 → 5 jobs of 1000 events
 ├── Analysis Stage
 │    ├── Aggregate results (per bin, pion/kaon)
 └── Final Stage
      └── Compute objectives & feed Ax
```



Case study 2: dRICH-MOBO optimization

- Performance against different number of events and iterations
- Scaling and Overheads
  - Concurrent simulation and evaluation
  - Trial generation overheads
  - Scheduling waiting for busy computing resources





Elapsed Time vs. Trial index of a dRICH-MOBO optimization with 5000 events

Performance Insights

- Efficient scaling across distributed resources
- Overheads from trial generation and scheduling latency (waiting for busy computing resource)
- Concurrent trial evaluation and simulation
- Automatic checkpoint and restart for long runs
- Optimization metrics are saved for later evaluation

Conclusion and Outlook

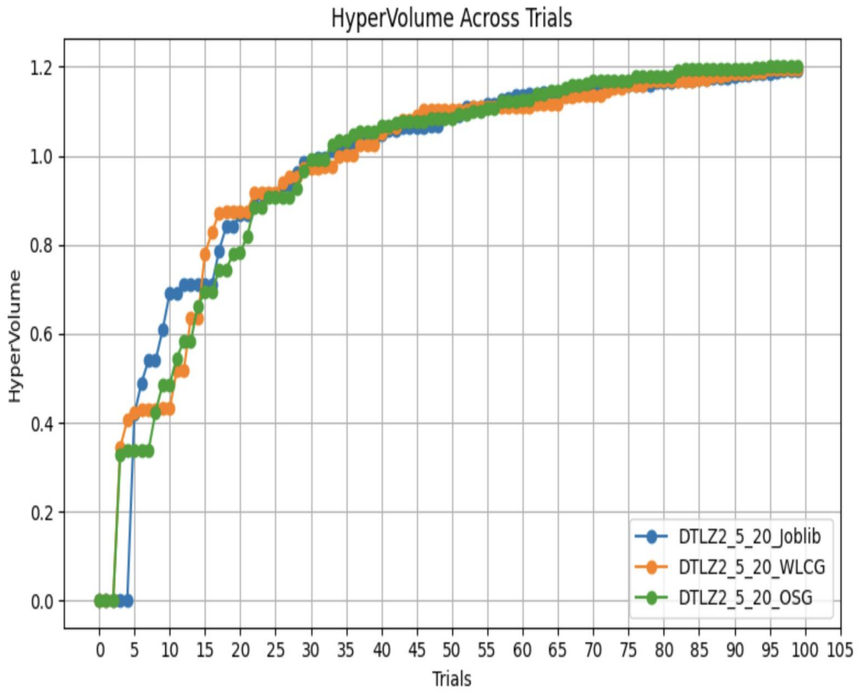
- PanDA/iDDS delivers flexible, scalable workflows for AI-driven science, enabling distributed HyperParameter Optimization and Active Learning
- AID2E integration bridges ML optimization with distributed resources, accelerating ePIC detector R&D
- Successfully built an infrastructure to optimize the ePIC sub-detector systems, though a lot of open design questions remain open and continue to drive further exploration
- Next steps:
  - Extending to large ML models and more detector components
  - Explore integration with Large Language Models (LLMs) to enhance automation, workflow orchestration, and adaptive decision-making

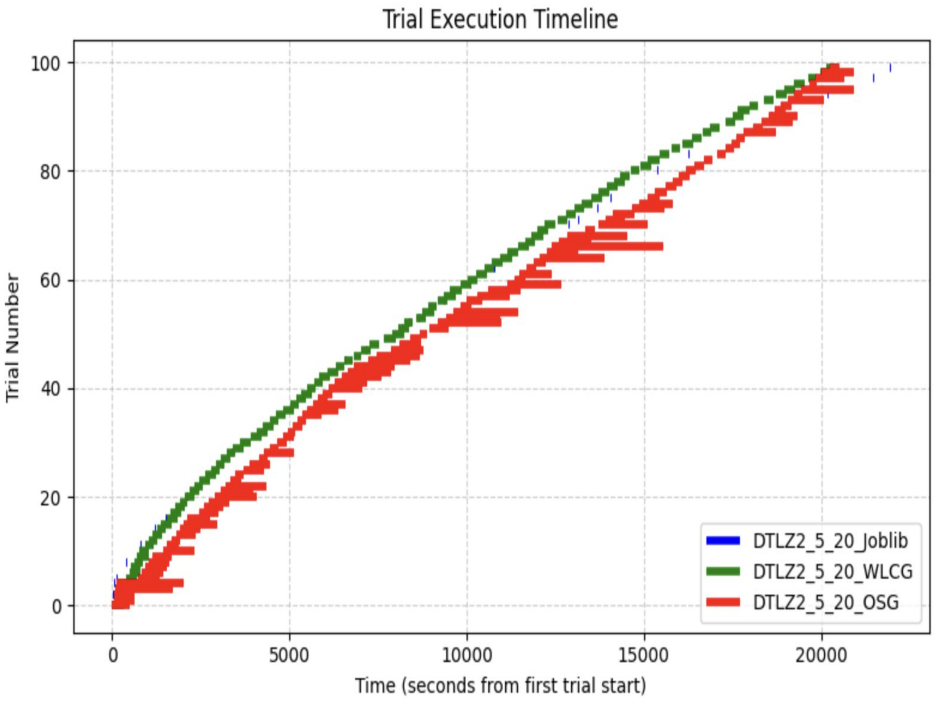
Backups

Backup: DTLZ2

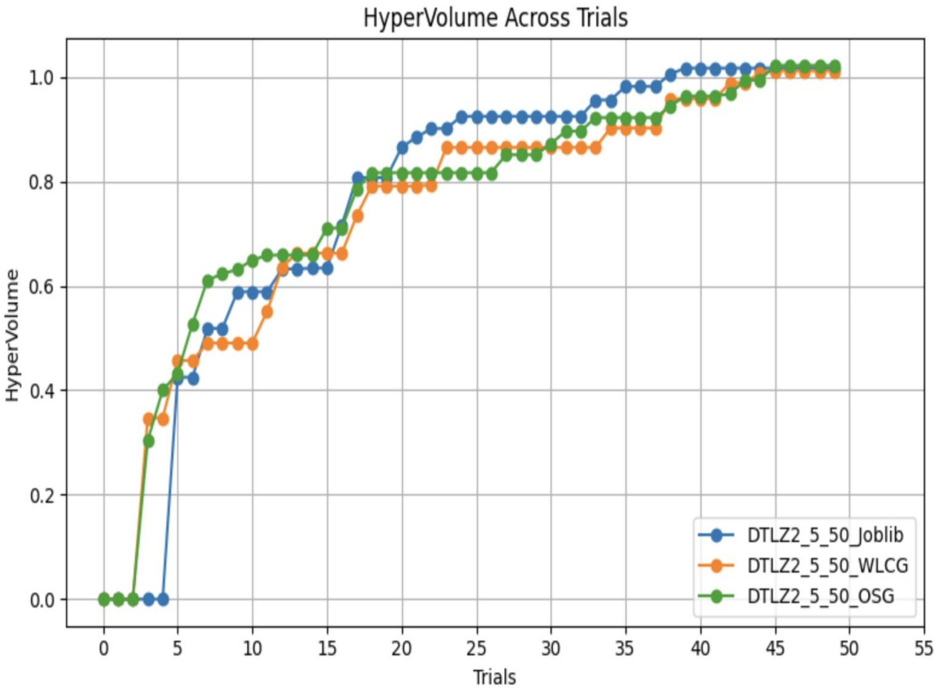
- JobLib(local): single process
- Max concurrent trails = 5 for PanDA
- 5 objectives, 20 parameters:
  - Trial generation is fast. Execution times are comparable between Joblib (local) and PanDA running on WLCG/OSG.
- 5 objectives, 50 parameters:
  - Trial generation becomes slower, and parallel efficiency decreases.
  - PanDA (WLCG/OSG) shows longer runtimes compared to local execution.
- 5 objectives, 100 parameters:
  - Trial generation is significantly slower, with further reduced parallelism.
  - PanDA on WLCG/OSG exhibits the longest execution times due to higher parameter dimensionality and scheduling overhead.

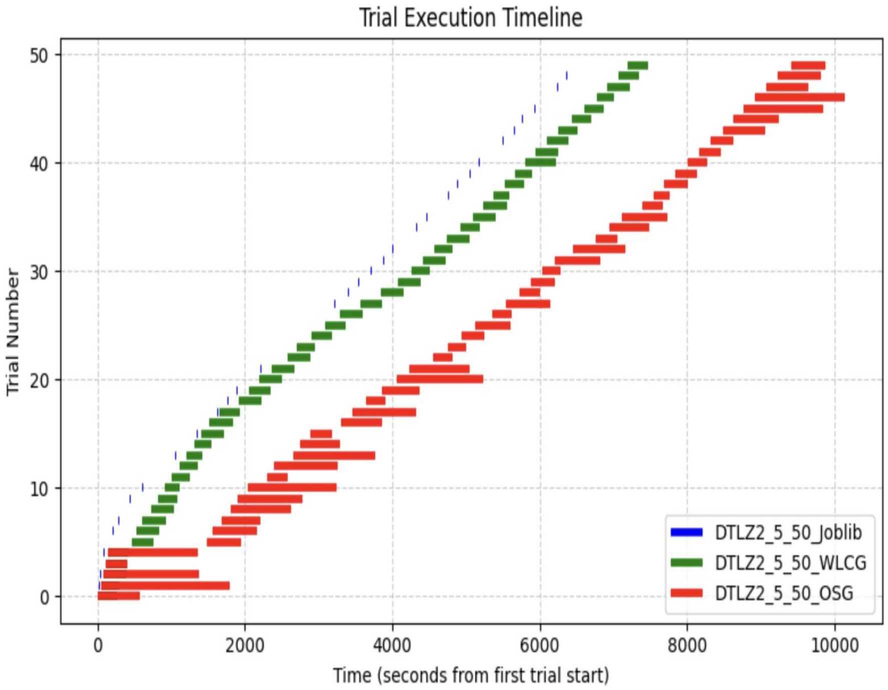
DTLZ2: 5 objectives 20 parameters



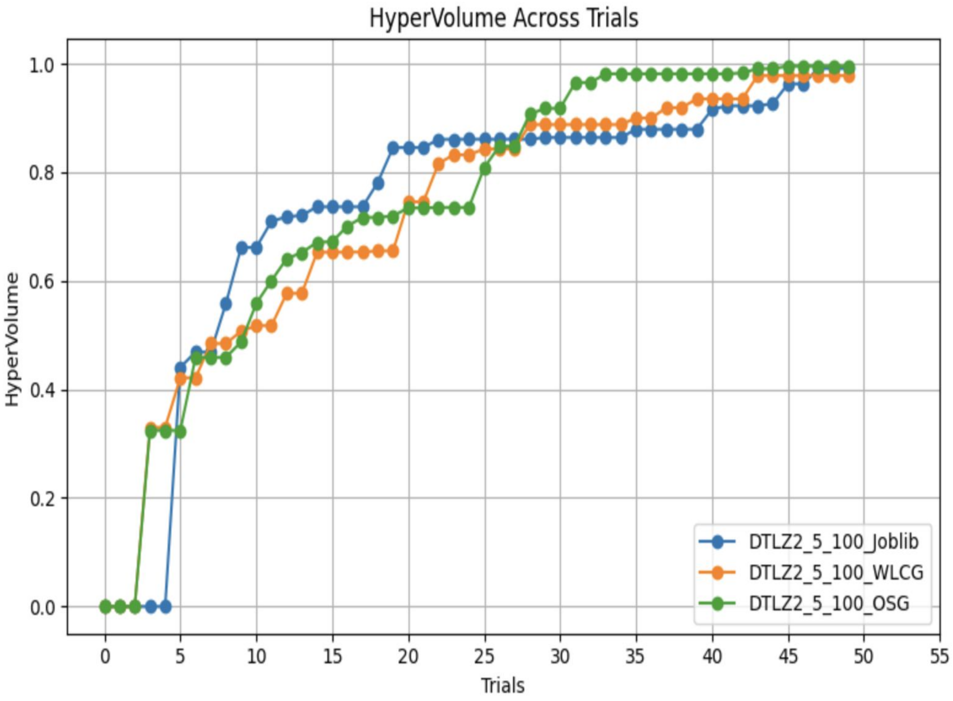


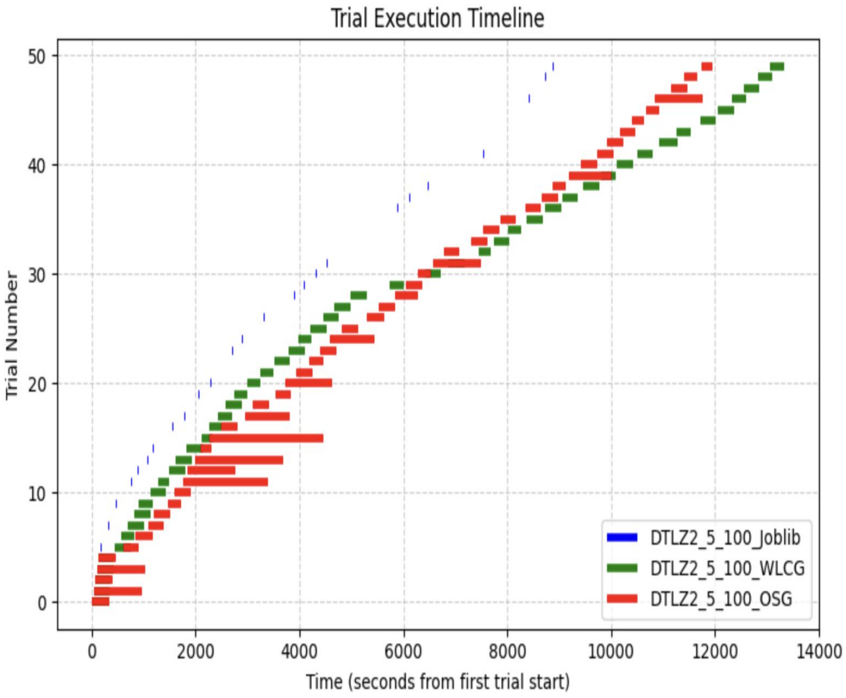
DTLZ2: 5 objectives 50 parameters





DTLZ2: 5 objectives 100 parameters





Function-as-a-Task

- **Python function as a task/job**
  - With python decorator to convert python functions to PanDA tasks/jobs to be executed at remote sites
  - Automatically transfer the function outputs back
  - Transparent to users like a local function
  - More granular for scientific workloads
  - Complicated logics can be defined with python language
  - Easy for users with python to define workloads
  - Support list of function parameters to map functions to multiple concurrent jobs
- **Asynchronous retrieval of function results**
  - Employ messaging service to publish/receive results between function executor and submitter
  - Fallback to Rest services if the messaging service is not accessible.

```python
@work(map_results=True)
def optimize_work(opt_params):
```

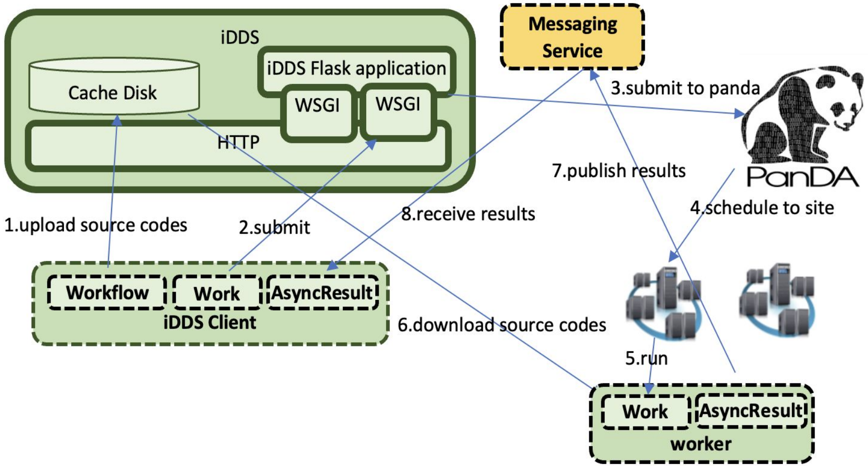
With python decorator @work to convert a function to a PanDA task

```python
@workflow
def optimize_workflow():
    from optimize import evaluate_bdt, get_bayesian_optimizer_and_util

    ...
    n_iterations, n_points_per_iteration = 10, 20
    for i in range(n_iterations):
        points = {}
        group_kwargs = []
        for j in range(n_points_per_iteration):
            x_probe = bayesopt.suggest(util)
            u_id = get_unique_id_for_dict(x_probe)
            print('x_probe (%s): %s' % (u_id, x_probe))
            points[u_id] = {'kwargs': x_probe}
            group_kwargs.append(x_probe)

        results = optimize_work(opt_params=params, group_kwargs=group_kwargs)
        print("points: %s" % str(points))
```

The Workflow calls the task like a local function

Function-as-a-Task Schema

- **Environment preparing**
  - Source codes caching
    - workflow as the basic unit to manage source codes
    - Source codes in the workflow directory will be uploaded into the iDDS or PanDA http cache
    - During running time, the source codes will be downloaded to the current running directory
  - Running environment
    - Base environment (eg: cvmfs) + source codes caching
    - Base container + source codes caching
- **Workload**
  - Submit function as tasks/jobs to workload management system PanDA
  - Load and run a function as a job at distributed sites
  - List of parameters can be used to call a function, which will create a task with multiple jobs and every job uses item of the list of parameters
- **Results retrieval**
  - When a function finishes, the executor will publish the result in a message
  - The submitter will receive the result
- **iDDS also monitors the tasks/jobs submitted. It will publish messages to AsyncResult, to avoid AsyncResult waiting for failed remote workers**



Schema of how a workflow executes a function at remote distributed resources

Function-as-a-Task Advantages

- Source codes are managed transparently, no additional steps
- Support different ways to run user functions at distributed resources
  - With/without container
  - With base container + source codes caching, users don't need to build the container for a code update, the workflow will automatically update the source codes in the cache
  - For some experiments, different base containers are already provided and deployed on cvmfs. Users don't need to build personal containers
- Make use of the current PanDA structure and related middlewares, no additional requirements for sites
- Distributed resources, possible to large scale
- Asynchronous results retrieval based on messaging service improves the efficiency

Thanks