Yoda

Event Service Implementation for HPC (Concept)

Vakho Tsulaia

LBL

US ATLAS S&C Meeting
Berkeley, August 21, 2014

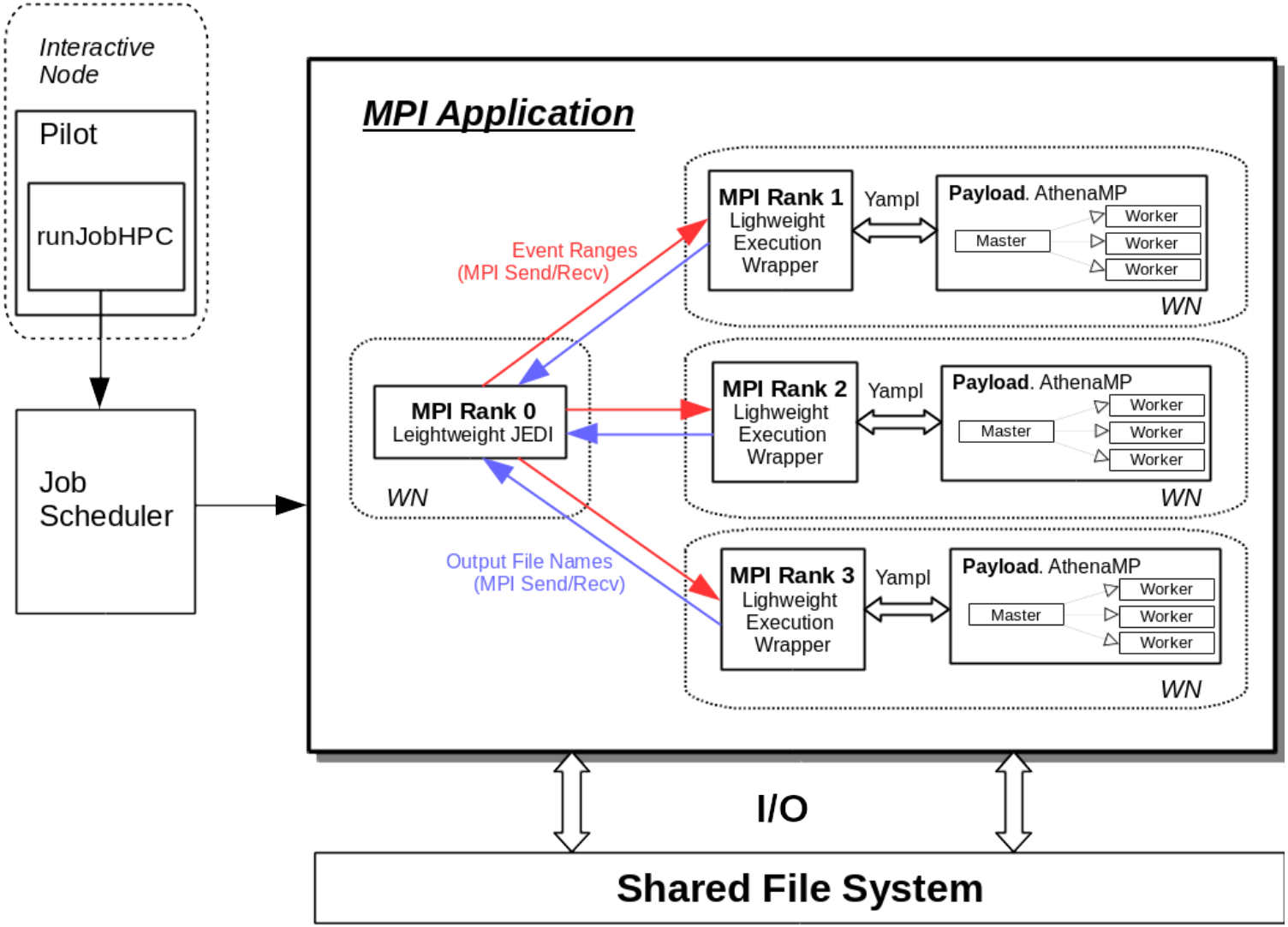
BERKELEY LAB

Introduction

- On (most) HPCs the architecture and accessibility limitations make **operation of the conventional Event Service impossible**
  - No outbound internet connection from the compute nodes
- For such architectures the Event Service functionality needs to be implemented in a **new way**
- **Proposal:** turn Event Service into a **MPI-application**
  - **Rank 0:** a lightweight JEDI
  - **Rank N** (N!=0): a lightweight Pilot/Execution Wrapper
- Name for such MPI-application: ***Yoda***

Design

Document: https://twiki.cern.ch/twiki/pub/PanDA/EventServer/Yoda.pdf



Running

- Yoda job should be submitted to the batch system like a “regular MPI” job
- Example (using **`aprun`** command):

```
aprun -n X -N 1 -cc none yoda.py [input parameters]
```

  - **X** (>1) here is the number of MPI-ranks for the given job
  - **-cc none** is used to avoid pinning all forked sub-processes to the same CPU core
- Skeleton for **`yoda.py`**:

```
from mpi4py import MPI
mpirank = MPI.COMM_WORLD.Get_rank()

if mpirank==0:
    # Run lightweight JEDI
else:
    # Run lightweight Pilot
```

MPI Ranks for Yoda

- **Rank 0** and **Rank N** in **Yoda** application perform basically the same tasks as **JEDI/PanDA Server** and **Pilot** in the **conventional Event Service**

- Thus, the idea is to **reuse the existing JEDI and Pilot code** for Yoda as much as possible
  - The complete functionality will not be necessary. That's why we are talking about **lightweight versions**

- One of the main difference:
  - **Conventional ES:** JEDI and Pilot communicate over **HTTP**
  - **Yoda:** JEDI and Pilot communicate using **MPI point-to-point communication mechanisms**

- **No changes are expected either for AthenaMP payload, or for Token Extractor**

Input

- All input files (EVGEN for G4Atlas) need to be available for Yoda on the **shared FS**
- In addition to that, for **each input EVGEN file** we need to make a **TAG file**
  - **Token Extractors** will use TAG files for Event Number to POOL Token conversion
  - In the absence of the outbound internet connection from the compute nodes, we **cannot use the Event Index**
- And, we also need to make ASCII file containing **EVGEN File GUID to TAG File Name mapping**
  - The same mapping files are used for the conventional ES
- The TAG files as well as the mapping files also must be accessible on the shared FS

Output

- AthenaMP writes the output files (one per each event range) **directly to the shared FS** and reports their location to the Pilot (Rank N)
- The Pilot passes this information over to JEDI (Rank 0)
- Rank 0 has several options for merging the outputs
  - **Initiate merging during the execution of Yoda**
  - Collect all info required for merging and pass it over to the job submitter application, which can proceed with merging after the Yoda job has finished
  - **Follow the approach of the conventional ES and upload the outputs to an external aggregation point (Object Store)**

Monitoring

- Rank 0 will use **SQLite** files for storing **Event Table** and **Job Table** for the Yoda job
- The SQLite databases will also be available on the shared FS
- The information from these SQLite files can be **passed outside of HPC** to the **central PanDA services** for external monitoring of the running Yoda jobs