Pilot Software in the SPD Online Filter

Romanychev Leonid
JINR MLIT
romanychev@jinr.ru

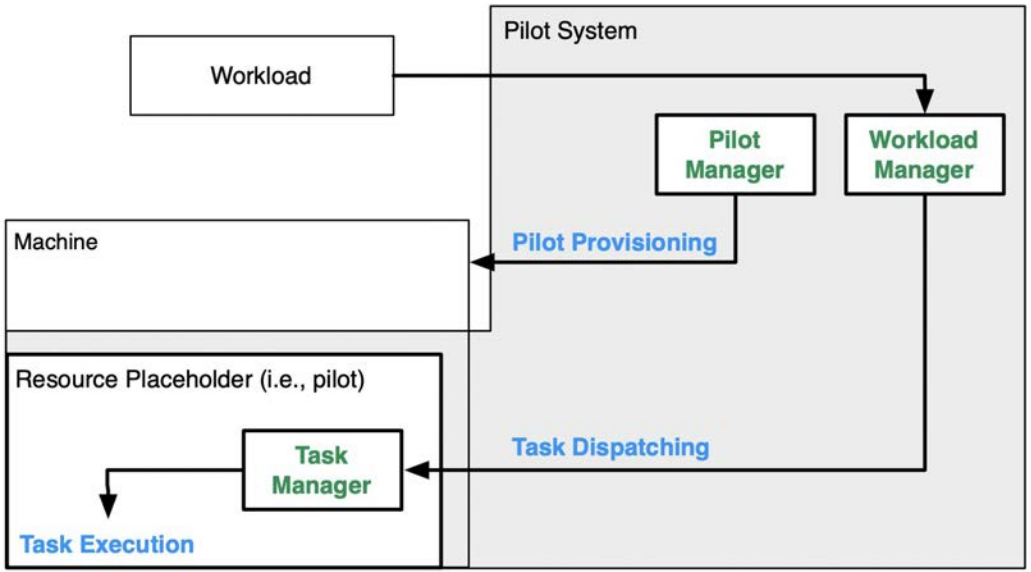
JINR AYSS Conference “Alushta-2025”
11.06.2025

Introduction: Role of Pilot Applications

- Provide a flexible mechanism for execution of computational tasks.
- Widely used in high-throughput computing (HTC) systems for scientific data processing (ex. LHC computing).
- Issue: lack of unified abstraction and best practices leads to a variety of implementations.

Core Components of Pilot Software

- **Pilot Manager:** Launches pilots (resource placeholders) on computing resource; Interfaces with SLURM, HTCondor, etc.
- **Workload Manager:** Organizes task queue (dependencies, priorities, resource readiness).
- **Task Manager:** Executes tasks on pilot-reserved resources; Manages task lifecycle (launch, restart, monitor, error handling).



Functionality & Architecture

Functional Stages:

- Provisioning (Acquiring & deploying resources)
- Dispatching (Assigning tasks to pilots)
- Execution (Running tasks on resources)

Architectural Features:

- Multi-level scheduling
- Communication Models (Master-worker, Broker-oriented)
- Flexibility (integrates with various DCRs: clusters, grids, clouds)

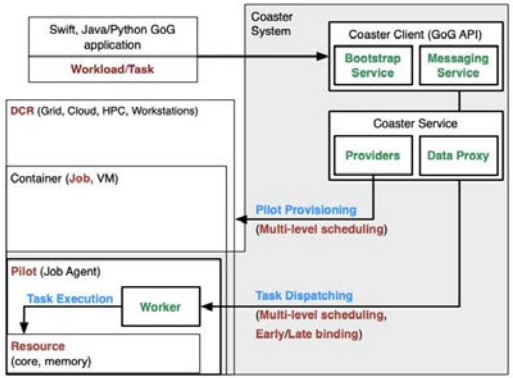
Late Binding Mechanism

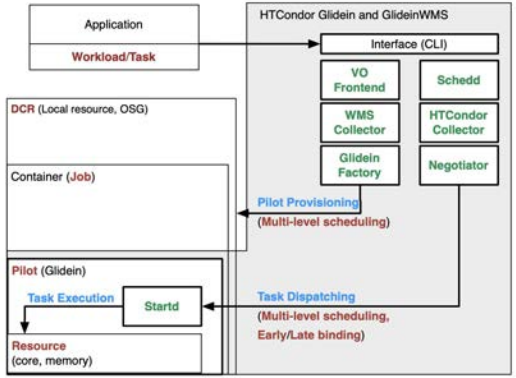
Definition:

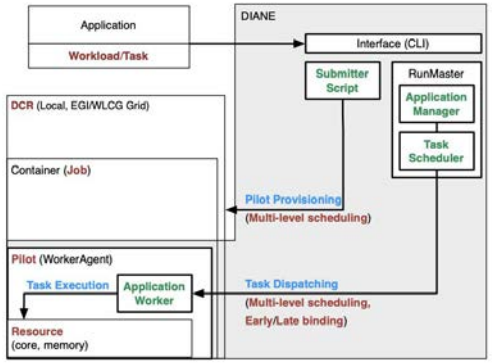
Late binding is the process of assigning tasks to active pilots at the moment of availability, unlike early binding, where tasks are tied to inactive pilots.

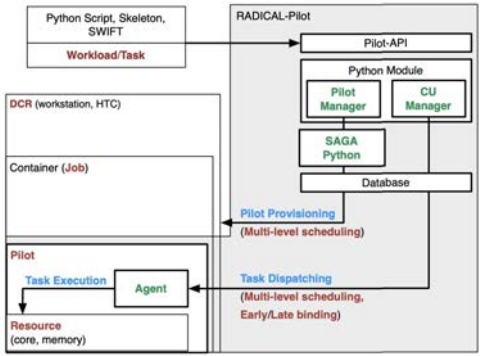
Benefits:

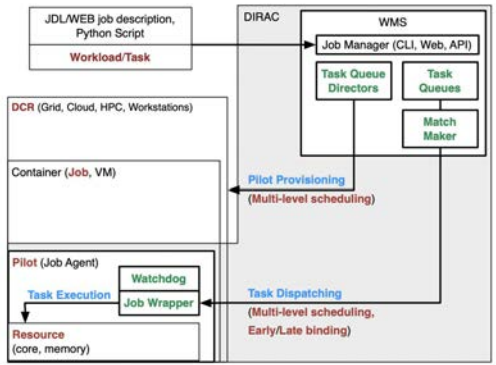
- Dynamic task allocation improves resource utilization efficiency.
- Reduces queue wait times, critical for high-performance systems.
- Enables high throughput (e.g., up to 1 million tasks per day for ATLAS).

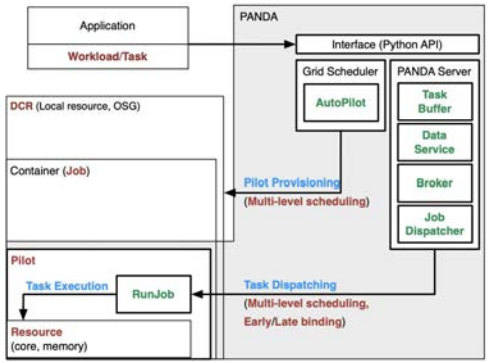

Swift, Java/Python GoG application
Workload/Task
Coaster System
Coaster Client (GoG API)
Bootstrap Service
Messaging Service
Coaster Service
Providers
Data Proxy
DCR (Grid, Cloud, HPC, Workstations)
Container (Job, VM)
Pilot Provisioning
(Multi-level scheduling)
Pilot (Job Agent)
Task Execution
Worker
Task Dispatching
(Multi-level scheduling,
Early/Late binding)
Resource
(core, memory)


Application
Workload/Task
HTCondor Glidein and GlideinWMS
Interface (CLI)
VO Frontend
Schedd
WMS Collector
HTCondor Collector
Glidein Factory
Negotiator
DCR (Local resource, OSG)
Container (Job)
Pilot Provisioning
(Multi-level scheduling)
Pilot (Glidein)
Task Execution
Startd
Task Dispatching
(Multi-level scheduling,
Early/Late binding)
Resource
(core, memory)

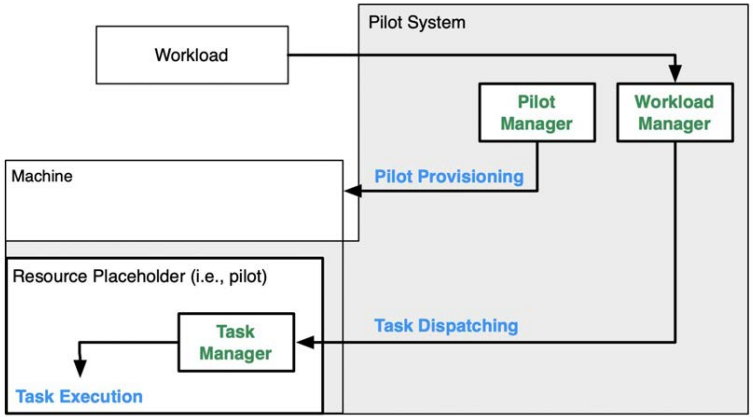

Application
Workload/Task
DIANE
Interface (CLI)
Submitter Script
RunMaster
Application Manager
Task Scheduler
DCR (Local, EGI/WLCG Grid)
Container (Job)
Pilot Provisioning
(Multi-level scheduling)
Pilot (WorkerAgent)
Task Execution
Application Worker
Task Dispatching
(Multi-level scheduling,
Early/Late binding)
Resource
(core, memory)

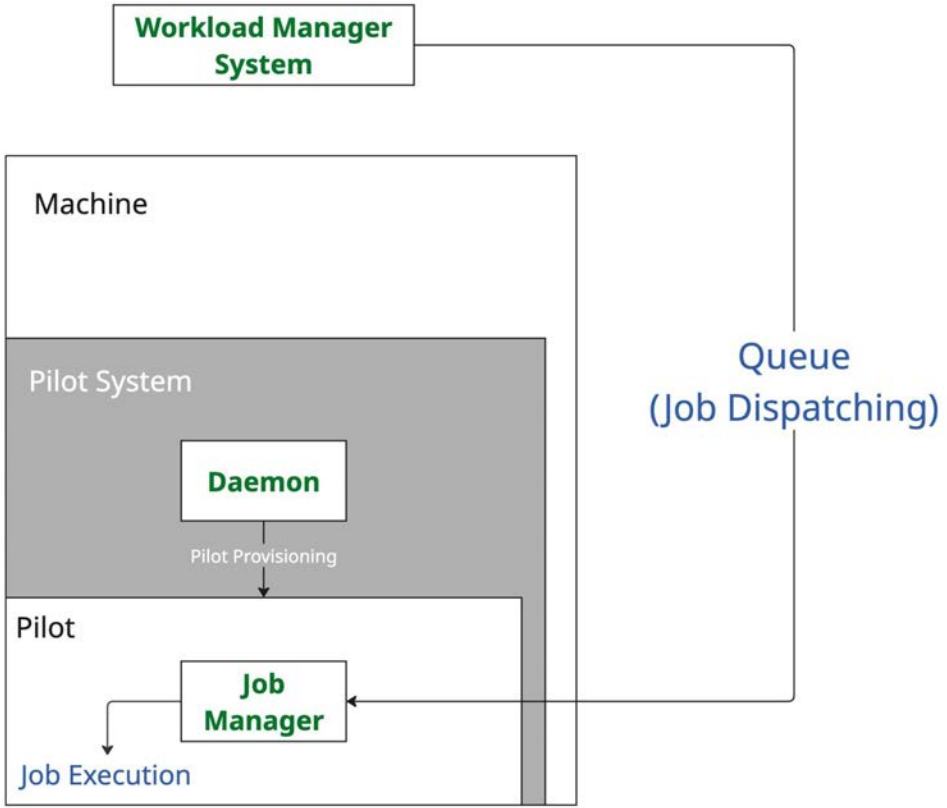

Python Script, Skeleton, SWIFT
Workload/Task
RADICAL-Pilot
Pilot-API
Python Module
Pilot Manager
CU Manager
SAGA Python
Database
DCR (workstation, HTC)
Container (Job)
Pilot Provisioning
(Multi-level scheduling)
Pilot
Task Execution
Agent
Task Dispatching
(Multi-level scheduling,
Early/Late binding)
Resource
(core, memory)


JDL/WEB job description, Python Script
Workload/Task
DIRAC
WMS
Job Manager (CLI, Web, API)
Task Queue Directors
Task Queues
Match Maker
DCR (Grid, Cloud, HPC, Workstations)
Container (Job, VM)
Pilot Provisioning
(Multi-level scheduling)
Pilot (Job Agent)
Watchdog
Task Execution
Job Wrapper
Task Dispatching
(Multi-level scheduling,
Early/Late binding)
Resource
(core, memory)

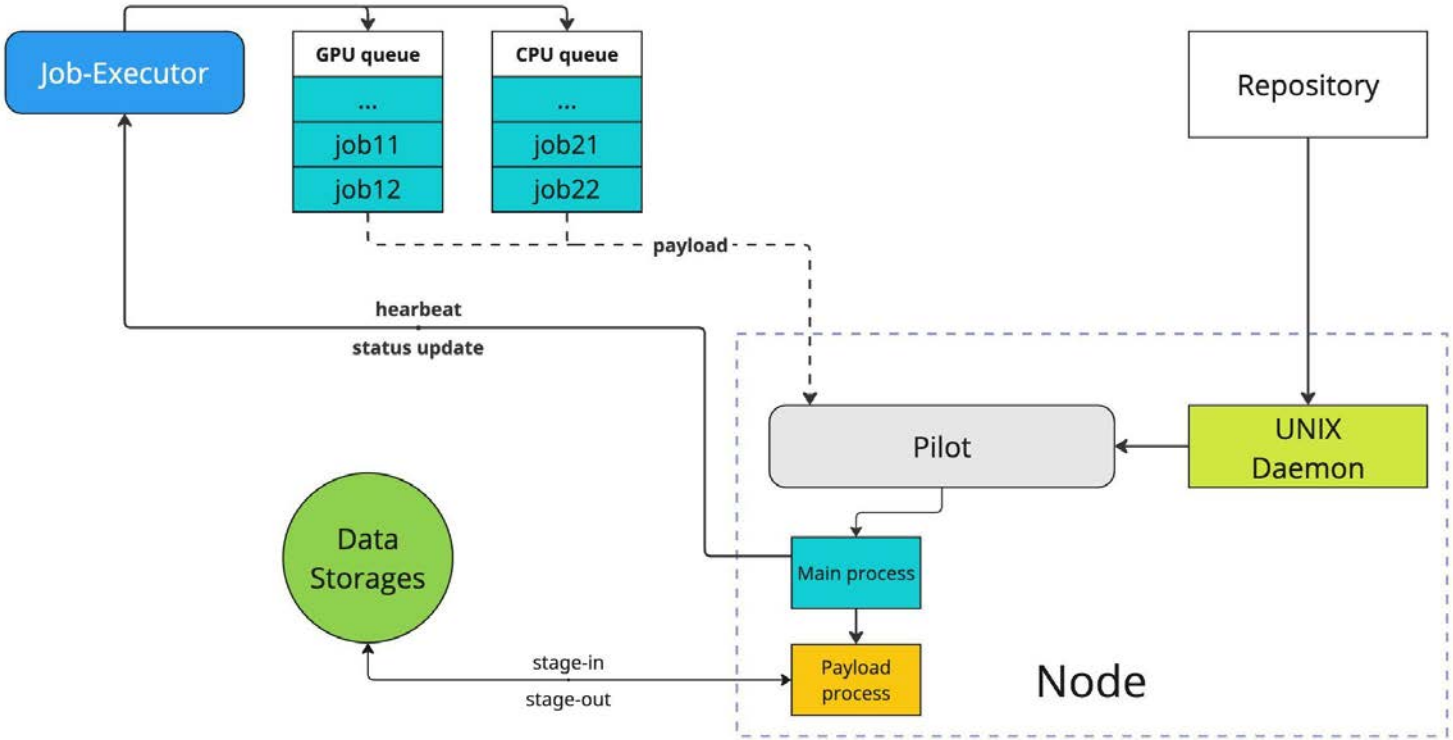

Application
Workload/Task
PANDA
Interface (Python API)
Grid Scheduler
AutoPilot
PANDA Server
Task Buffer
Data Service
Broker
Job Dispatcher
DCR (Local resource, OSG)
Container (Job)
Pilot Provisioning
(Multi-level scheduling)
Pilot
Task Execution
RunJob
Task Dispatching
(Multi-level scheduling,
Early/Late binding)
Resource
(core, memory)

SPD Online Filter Pilot Software

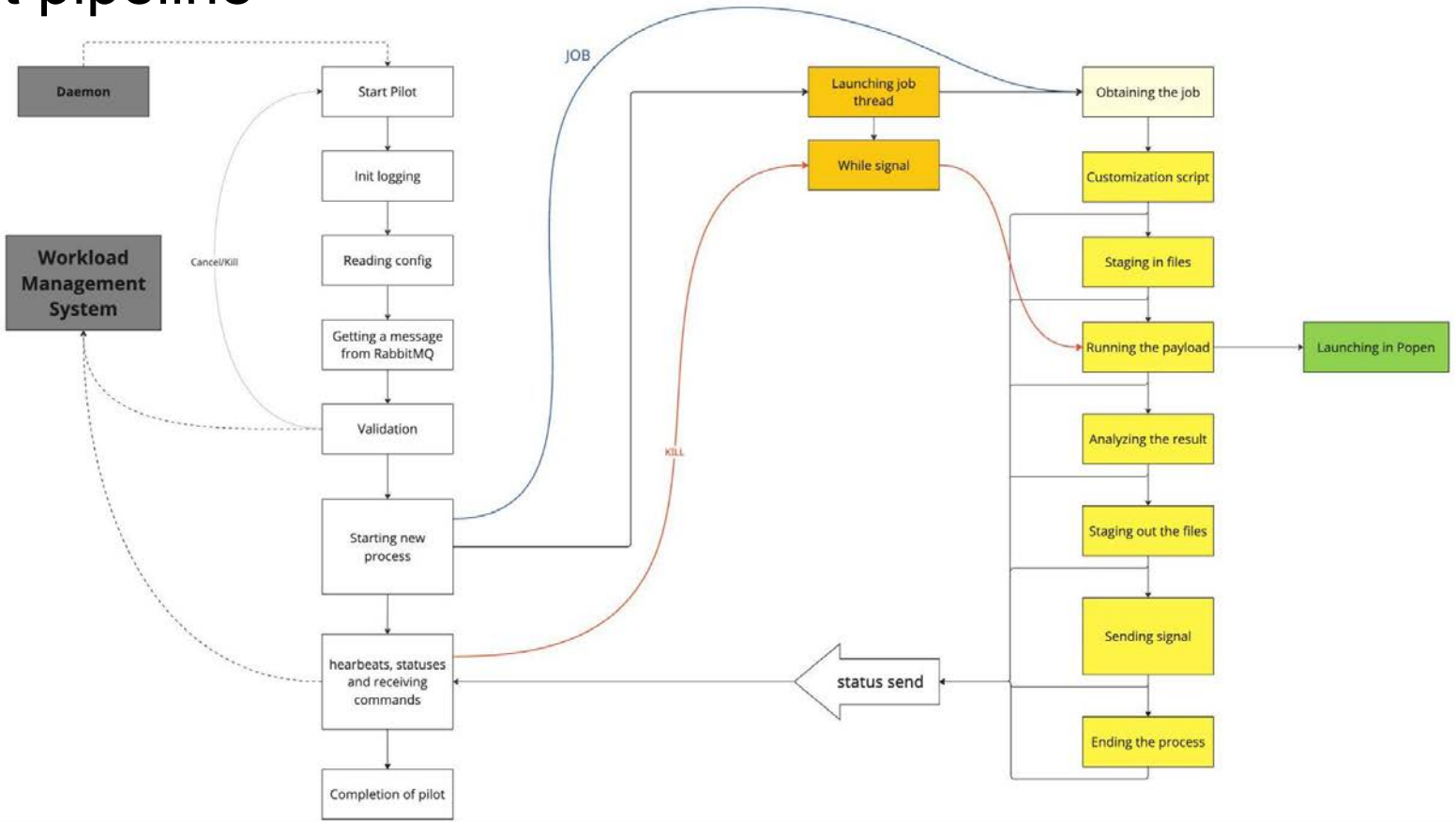




SPD Online Filter Pilot Software



Pilot pipeline



“DAQ emulator”

1. Using SPD DAQ emulator, we’ve generated 50 files, each ~2Gb;
2. Input dataset has been registered with these files;
3. Task has been processed (or 50 jobs);
4. The payload for Pilot is simple: compute the MD5/BLAKE3 hash, as there is no actual computation involved at this stage.;
5. Generation of one files takes around ~7 min, using JINR Cloud VM: 12x 1-core Intel Xeon E5-2650
6. Registration of the entire dataset: ~10 sec

```
# Configuration file for SPD DAQ data generator
# 2023/03/01

#Data file name format: run-<run number>-<chunk
number>-<builder id>.spd
DataFileNameFormat = run-%06u-%05u-%02u.spd

#RND generator seed:
RandomSeed = 12345

#The size limit of the output data file in bytes:
DataFileSizeLimit  = 2147483648

#debug mode for debuging front-end card. If it is 1
then generator will
#produce all data words (headers and trailers) even
if there are no hits,
#otherwise all empty data blocks are removing
DebugMode          = 0

#Source ID(s) of the clock modulue(s) for
measurement start of frame time:
FrameClockID       = 1000,1001

#Source ID(s) of the TDC module(s) for measurement
of the bunch crossing time:
BunchCrossingID    = 1004

#Slice length in ns (must be less than smallest TDC
over-roll time (4.5 ms for RS)):
SliceLength        = 10000

#Number of slices in a frame:
FrameLength        = 100000
```

Conclusion

- Unified task execution interface
- Adaptability across platforms
- Reduced overhead (scheduling & execution)
- Scalability (to millions of concurrent tasks)

Thanks for your attention