Harvester

Tadashi Maeno (BNL)

Outline

- Motivation
- Design
- Workflows
- Plans

Motivation 1/2

- PanDA currently relies on server-pilot paradigm
  - PanDA server maintains state and manages workflows with various granularities, such as task, job, and event
  - Pilots are job-centric and independently run on worker nodes (WNs) with limited view of local resource
- Works well for the grid with 250k cores 24x7 as underlying resources are not very heterogeneous
  - But missing capability to dynamically optimize resource allocation among queues (score, mcore, himem, long, ...)
- Not very well for HPC or large-scale clouds
  - Each HPC has a different edge service and operational policy, leading to over-stretched pilot architecture and incoherence in implementation at different HPCs
  - PanDA itself has no means of managing and monitoring cloud utilization by using native cloud API which is far more optimal than that of an intermediate service like condor

Motivation 2/2

- New model : server-harvester-pilot
  - Harvester is a resource-facing service between PanDA server and collection of pilots
  - Stateless service with knowledge of resource
  - Modular design for different resource types
  - Many harvester instances running in parallel
  - To provide a single view of a large or uniform resource that optimizes pilot and/or workload management
  - To provide a commonality layer in bringing coherence to HPC implementations
  - Better integration with PanDA system for various (new) workflows, such as job/event-level late-binding and jumbo jobs

Design 1/2

- Details
  https://docs.google.com/document/d/1zVj9cSrFbWf7HRaqwr66yOfvJpit8VmiwGUpF5xQ4vU/edit?usp=sharing
- Key points
  - Lightweigh
    - To run on logon/edge nodes at HPC centers
  - Stateless for scalability + central database (oracle) + local database (sqlite3)
    - Capability to rebuild the local database from the central database for auto restart
    - Local database to reduce redundant access to the central database
    - Only important checkpoints are propagated to the central database
  - Installation with or without root privilege

Design 2/2

- Key points (cntd)
  - Configurability
    - To customize workflow for each resource
    - To turn on/off components with various plugins
  - Running on top of pilot API
    - Core + plugins + resource specifics in resource managers or pilot components
    - Leveraging development effort for the pilot consistently with the evolution plan (pilot 2.0)
  - Direct bi-directional communication with PanDA
    - Requesting workload to PanDA based on dynamic resource availability information and static configuration
    - Receiving commands directly from PanDA to throttle or boost the number of workers (worker = pilot, MPI job, or VM)

Workflows 1/3

- Workflow example for HPC + network-less WNs + conventional jobs + multiple workers (one PanDA job × one worker)
  1. Fetch PanDA jobs
  2. Stage-in input files
  3. Check with the batch system to find available slots
  4. Make MPI jobs from PanDA jobs
  5. Submit MPI jobs
  6. Monitor MPI jobs and update PanDA job status
  7. Stage-out output and log files
  8. Send final heartbeats

Workflows 2/3

- Workflow example for cloud + multiple workers (one PanDA job × one worker)
  1. Request large workload to be assigned
  2. Make a vanila image with the pilot
  3. Spin up VMs with the image using cloud API
  4. Send PanDA secret key(s) to them via contextualization

  Then the pilot takes care of subsequent procedures such as getting/updating job and stage-in/out

Workflows 3/3

- Job level late-binding on HPC
  - Currently for HPC, PanDA jobs are embedded into workers (MPI jobs), workers are pushed to the batch system, and then they get started once CPUs become available
    - Pre-binding of workload. I.e. cannot be reassigned even if other resources become free
    - Long latency if workers are in the batch queue for a long time (e.g., a few days of latency on busy HPCs like NERSC), which is problematic for high priority tasks
  - Synchronized communication is required between Harvester and workers
    - Feeding PanDA jobs to workers once they get CPUs
    - Implementation details in the google doc

Jumbo Jobs 1/4

- New workflow. Details
  https://docs.google.com/document/d/11Xw4ee0VsxCyVKaxYoehu4xUHwomP6PO7zn7BWjAnc4/edit?usp=sharing
- HPC prefers large jobs to process a large number of events in one go
- Possible to configure tasks to generate large jobs, but
  - Difficult to run large jobs on traditional resources due to limited number of cores and walltime per job, so that those tasks can run only on HPCs and total amount of CPU resources are limited
  - Large jobs produce gigantic files since currently in PanDA one job produces one file for each data type like AOD and ESD
  - Gigantic files are not good if they are used by subsequent tasks since not all production steps use direct IO when reading files
  - Mixture of small and gigantic files in a dataset makes subsequent tasks complicated when the dataset is used as input, e.g. for pileup

Jumbo Jobs 2/4

- Goal
  - A machinery to dynamically tailor workload based on available CPU cores for each resource and at the same time to have a constant or similar number of events in each output file
- Implementation
  - A single jumbo job can process all events in a task
  - One or more jumbo jobs are defined for one task to be assigned to HPC sites
  - At the same time, normal jobs (co-jumbo jobs) are defined with reasonable number of events per job (e.g. 1000 events per job) to be assigned to conventional grid sites
  - Workers run jobs, and get events once they get CPUs

Jumbo Jobs 3/4

- Implementation (cntd)
  - Each event can be processed by one worker either with jumbo job or co-jumbo job, but once the event is processed by a worker other workers will not process the event again
  - Tasks can progress even if HPC workers are waiting in long queues since co-jumbo jobs can process events meanwhile
  - Each co-jumbo job generates output file(s) once all events assigned to the job are processed

Jumbo Jobs 4/4

- Status
  - Server side functions have been implemented
  - Some thoughts are still required for accounting and logs
  - No pilot scheduler or HPC pilots support jumbo jobs yet
- Jumbo jobs + late-binding
  - Jumbo jobs can mitigate the issue with pre-binding of workload due to co-jumbo jobs
  - Late-binding is still useful with jumbo jobs since high priority jobs can jump over low priority jobs
  - HPC could be well integrated in normal production activities using jumbo jobs + late-binding
- Jumbo jobs for cloud
  - Jumbo jobs are also useful for large-scale provisioning of workers on clouds since a single jumbo job can be processed by multiple workers

Plans

- Manpower
  - Core components : Tadashi + Fernando
  - Plugins for cloud : Fernando
  - Plugins for HPC : Experts for each HPC workflow from pilot team
- First focus on HPC and large-scale cloud
- Prototype
  - Core components with limited functions and one set of concrete plugins
  - The primary target is HPC (Titan or NERSC) + network-less WNs + multiple jumbo jobs
    - Not a toy but beneficial to bring HPC resources to the normal production without custom tasks or human interventions
  - Timeline
    - ~ after CHEP
- Github repository
  - https://github.com/PanDAWMS/panda-harvester
- Start with pilot 1.0 and then refactor plugins with pilot 2.0