Fine grained event processing using an Event Service

Vakho Tsulaia (LBNL)
For the ATLAS Event Service Team

XRootD Workshop @ UCSD
January 27, 2015

Contents

- Event Service – a new approach to event processing
- Current implementation of the Event Service for running ATLAS Geant4 simulations
- Handling of event data and meta-data by the Event Service: current limitations and further developments
- Deployment of the Event Service to various platforms: *HPC*, *clouds*, *ATLAS@Home*, *'conventional resources'*

A New Approach to Event Processing

- A fine grained Event Service (ES)
  - ✓ Job granularity changes from files to individual events
  - ✓ Deliver only those events to a compute node, which will be processed there by the payload application
  - ✓ Don't stage in entire input files
- ES is agile and efficient in exploring diverse, distributed, potentially short-lived (opportunistic) resources
  - ✓ HPCs, spot market clouds, volunteer computing
- Minimize use of costly storage in favor of strongly leveraging powerful networks
- The job runs either until it reaches its lifetime or until it gets terminated
  - ✓ Minimal data losses

A New Approach to Event Processing (2)

- Event Service is our approach to running event processing jobs on **opportunistic resources.** Common characteristics to using such resources
  - ✓ Quick start when they appear, quick exit when they are about to disappear
  - ✓ Robust against their disappearance with no notice: minimize losses
  - ✓ Use them until they disappear – soak up unused cycles
  - ✓ Fill them with fine-grained workloads: **send a steady stream of events and return outputs in a steady stream**

- Managers of **'conventional' resources** – especially VM/cloud based – love the idea of workloads that can be instantaneously jettisoned with negligible losses

- Data intensive, network centric, platform agnostic computing. **Applicable to any workflow that can support fine grained partitioning of the processing and its outputs**

ES Collaboration

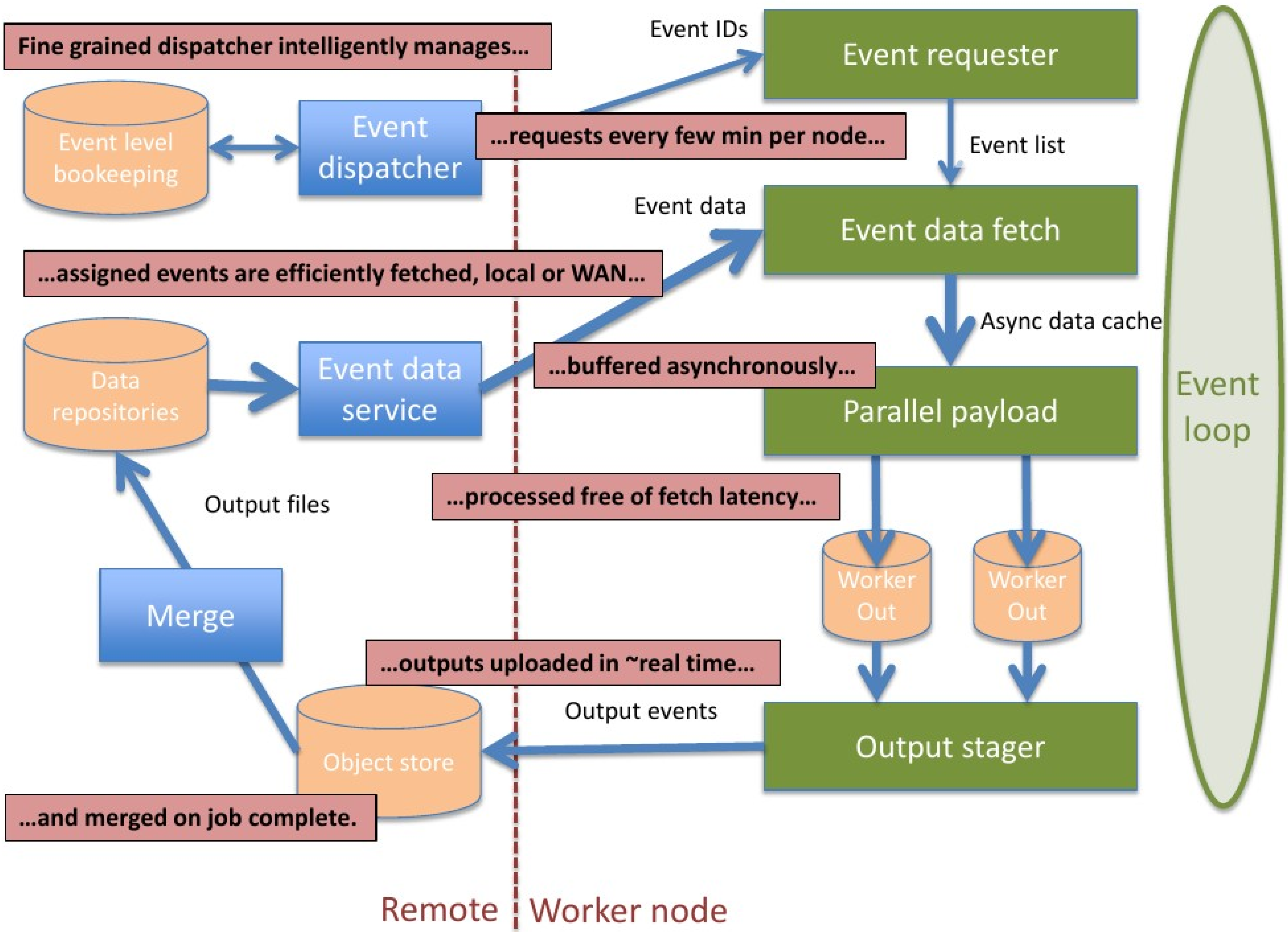
A broad Lab/University collaboration bringing together experts from various domains of ATLAS Computing: software development, distributed operations.

- BNL: (Big)PanDA and its JEDI fine grained extension, HPC porting
- LBNL: AthenaMP parallel processing framework, HPC porting
- ANL: Parallel I/O, WAN data access
- UTA: (Big)PanDA, HPC porting
- University of Wisconsin: Pilot, HPC porting

ES Components

- PanDA/JEDI
  - Job brokerage, workload management, bookkeeping
- AthenaMP (multi-process version of Athena – ATLAS reconstruction, simulation and data analysis framework)
  - Efficient usage of the CPU and memory resources on the compute node
  - Configured to process fine grained workloads (events, event ranges)
- Remote I/O
  - Efficient delivery of the event data to compute nodes
- Object store
  - Efficient management of the outputs produced by ES jobs

ES schematic

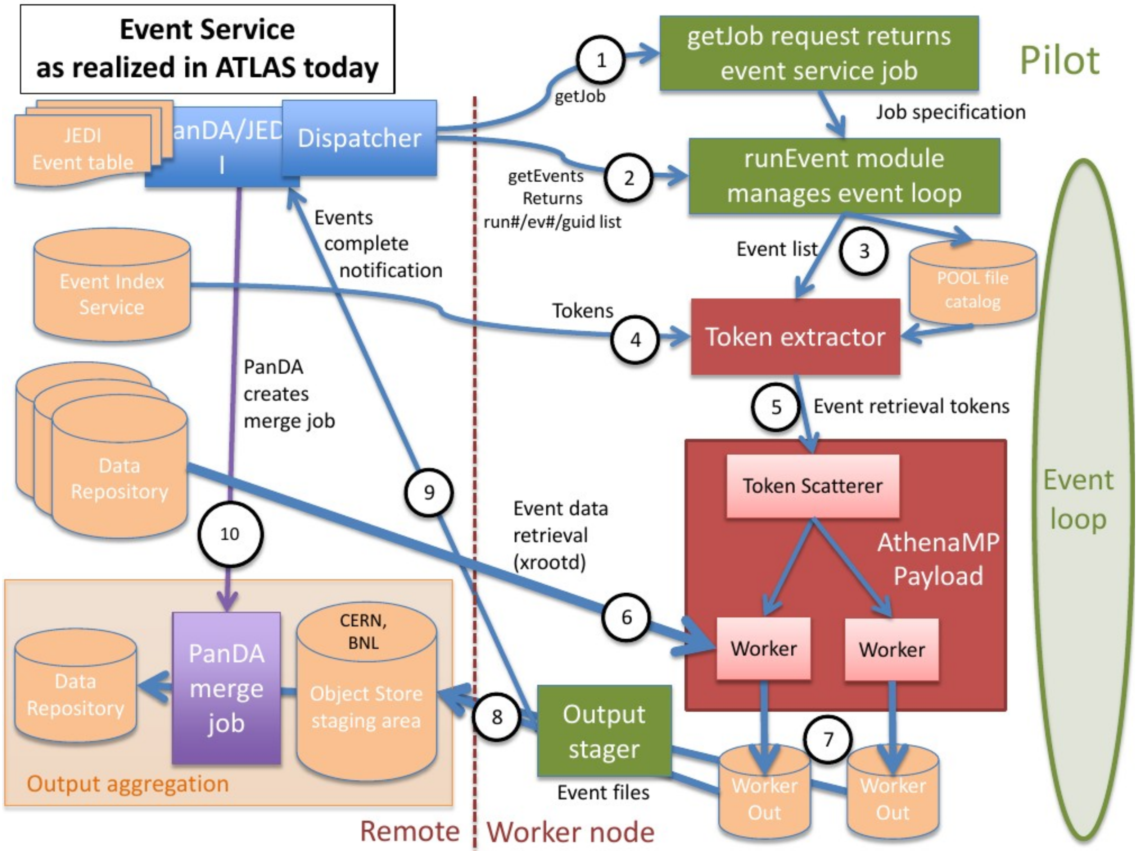


First use-case

- First implementation of the Event Service can run only **ATLAS Geant4 simulation**
  - ✓ The biggest return for the least investment
  - ✓ CPU-intensive job (**5-10min/event wall time**) with minimal I/O requirements (**<3MB/event output size**)
  - ✓ Meta-data handling relatively simple (**wrt other payloads**)

- **Other payloads expected to follow ...**

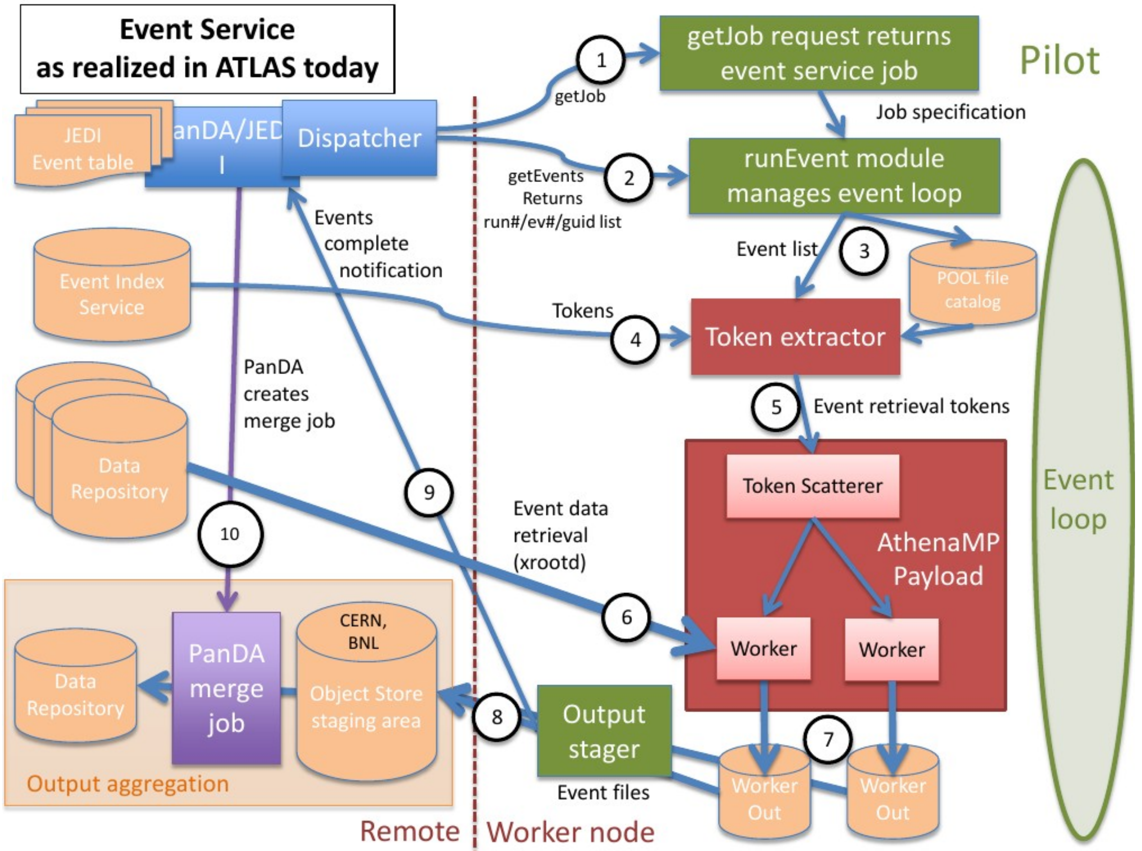
Current Implementation



- **Pilot** lands on an empty compute node and pulls a job definition from **PanDA/JEDI**
- Pilot receives a request to start an **ES Job** and launches the **payload AthenaMP** with no input
- AthenaMP goes through the initialization stage and informs Pilot that it is ready to start event processing
- Pilot starts pulling **ES work load** from PanDA/JEDI

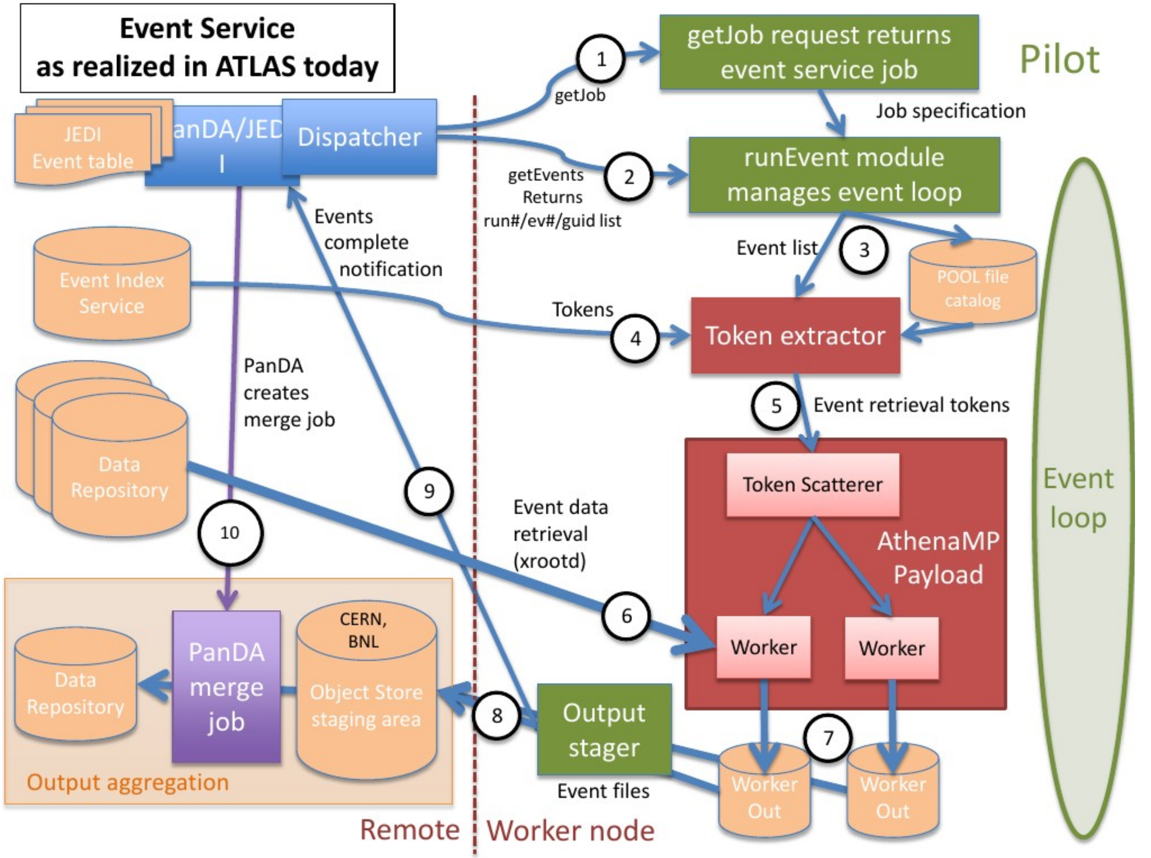
- Pilot gets the workload in the form of **Event Ranges** – strings, which contain: **Range ID, Input file name and ID, positional event numbers within the file (first event, last event)**

Current Implementation (2)



- AthenaMP transforms Event Ranges into the list of **Tokens**
- The list of Tokens ultimately gets delivered to **AthenaMP Worker Process**
- Each AthenaMP Worker retrieves **event data independently** using the Token and a local **File Catalog**
- AthenaMP worker creates **new output file for each Event Range**

Current Implementation (3)



- Pilot promptly streams the output files to **Object Stores** at BNL, CERN
- Panda **keeps event range statuses – running/completed/failed –** up to date in the database
- Failed event ranges are **re-dispatched** later on
- Finally the outputs are **merged**. PanDA triggers the start of a merge job

Inefficiencies of Event Data Reading

- Current implementation or event data reading in ES is rather **inefficient**
- Each worker process **reads event data separately** from other workers
- In the case of Geant4 Simulation, the worker usually needs only one event from a given ROOT basket. **The rest of the basket is discarded**
  - In our CPU-intensive simulation jobs 1 Event Range = 1 Event
- This results in **many duplicate transfers** of the same basket over the network to different worker processes
- This can be avoided by matching the number of events in the range to the number of events in the basket
  - Does not apply to G4 simulation, but can be achieved for other payloads

Shared Reader

- In order to implement **Event Data Service** depicted on page 7, it is essential to have a mechanism for **transferring Event Data Objects between processes**
  - ✓ Current implementation of ATLAS I/O and Persistency infrastructure does not provide such functionality
- First step in this direction would be to develop a **Shared Event Reader** (Event Source) for AthenaMP
  - ✓ Single place for reading input data objects and decompressing ROOT baskets
  - ✓ Sends individual events or event ranges to the workers
  - ✓ Has a potential of **asynchronous reading of input event data**
- Implementation of such shared reader is a fairly non-trivial task
  - ✓ <u>Relatively</u> simple to develop shared reader for Simulation jobs (compared to shared readers for other types of payload)

Shared Writer

- Current approach of the Event Service to dealing with output files:
  - ✓ Use a special **Output File Sequencer** mechanism for writing new output file for each event range
  - ✓ As a result, each worker process creates **many small output files**
- This mechanism has been working well for Geant4 simulation jobs so far, …
  - ✓ Good scaling is yet to be confirmed for HPCs with shared file systems
- … however, for I/O-intensive payloads we consider implementing **Shared Writer** (Event Sink) processes
  - Only one process/thread writing to the disk
  - Less files to merge at the end

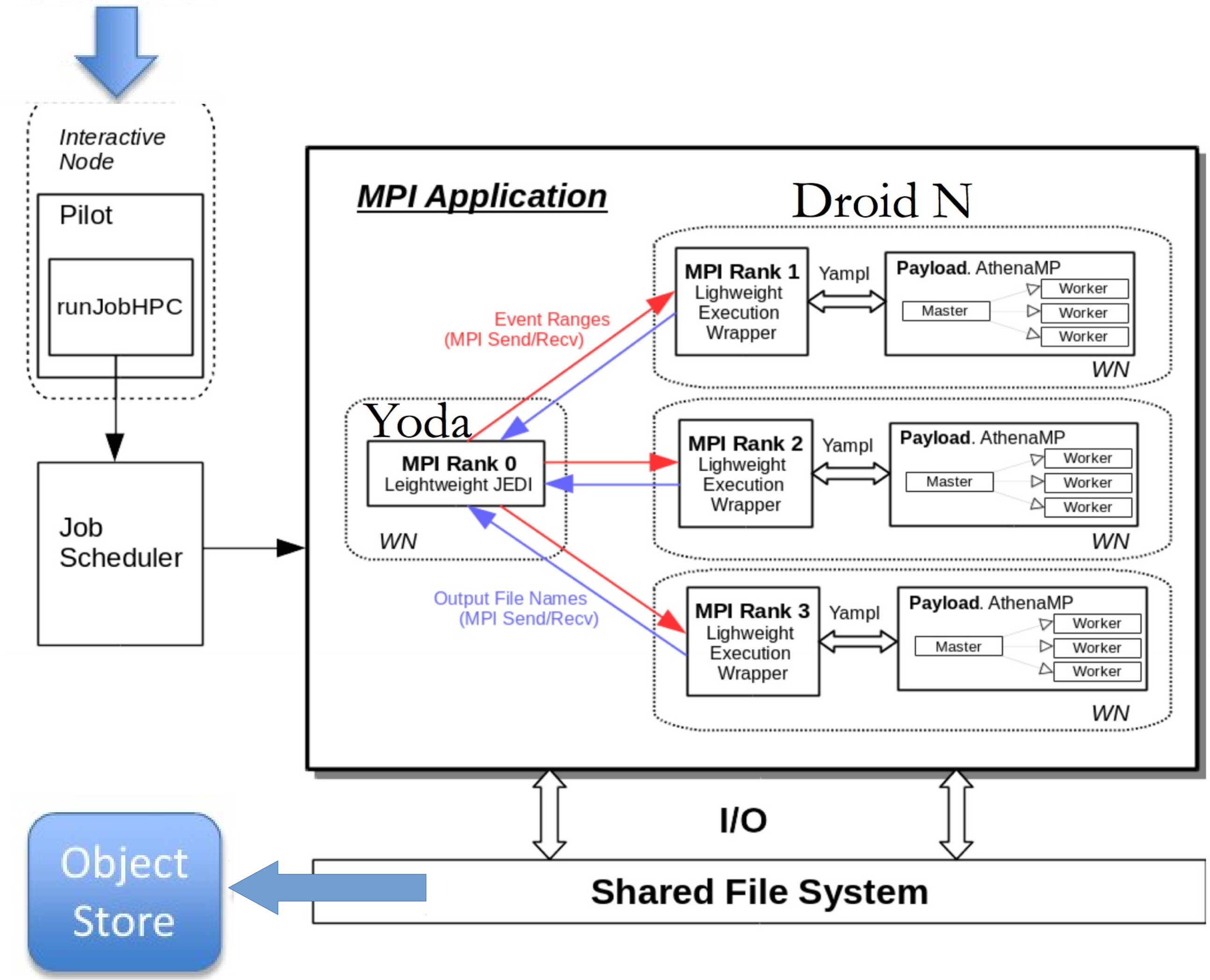
Handling of Meta-Data

- When switching from file-based to event-based workloads, the **handling of event meta-data is rather straightforward**
  - ✔ Events never span file boundaries
  - ✔ Event meta-data is written to the output right after processing the given event
- **In-file meta-data is not that simple**
  - ✔ Either accumulated/summarized over the run time of the job
  - ✔ Or propagated from the Input to Output file
- We store in-file meta-data with different purposes: *to be able to set up a job* (**I**nterval **O**f **V**alidity), *to describe a workflow, to describe the events*
- For working with **fine-grained workloads** – events, event ranges – **the existing meta-data infrastructure needs to be thoroughly changed/redesigned**
  - ✔ Relatively simple task for Geant4 simulation payloads

Deployment platforms

- Supercomputers
  - We have developed **Yoda** – a MPI-based implementation of the Event Service – specifically for running on HPCs
  - Yoda is flexible in defining duration and size of MPI jobs
  - Offers the efficiency and scheduling flexibility of **preemption** without the application needing to support or utilize checkpointing
  - By **reusing the code of the conventional event service**, we were able to very rapidly go from the concept of Yoda to its first implementation
  - Demoed at Supercomputing 2014 as a DOE ASCR Data Demo
  - **Currently being validated** by running ATLAS Geant4 simulation validation samples on **Edison supercomputer at NERSC (LBNL)**

Yoda



- **MPI application**
- **Reuse conventional ES code wherever possible**
- **Rank 0 (Yoda, master).** Distributes workload between slave ranks
- **Fine grained workload:** individual events or event ranges
- **Rank N (Droid, slave).** Processes assigned workload, saves output to the shared file system, asks for the next workload …

Deployment platforms (continued)

- Clouds
  - ✓ Event Service has been successfully tested on **Amazon Spot Market**
  - ✓ No scalability issues have been identified
- 'Conventional resources' (Grid)
  - ✓ Initial platform for the development and testing of the Event Service
  - ✓ Now we are about to **start Event Service commissioning on the Grid**
  - ✓ The idea is to get the ES up and running in production on the sites, which express interest in being its early adopters
- Volunteer computing (BOINC)
  - ✓ Underway …

Summary

- The concept of an Event Service is applicable to any workflow that can support fine grained partitioning of the processing and its outputs
- First implementation of the Event Service, which works only for Geant4 Simulation, is currently being validated
- We are ready to start Event Service commissioning on various deployment platforms: Grid, Clouds, HPCs
- Rapid progress so far, but lots of work ahead: the extension of the Event Service mechanism to other payloads beyond simulation is a major development task!