Event Service in ATLAS

Progress, problems and challenges

Vakho Tsulaia (LBNL)

For the ATLAS Event Service Team

U.S. DEPARTMENT OF ENERGY | Office of Science

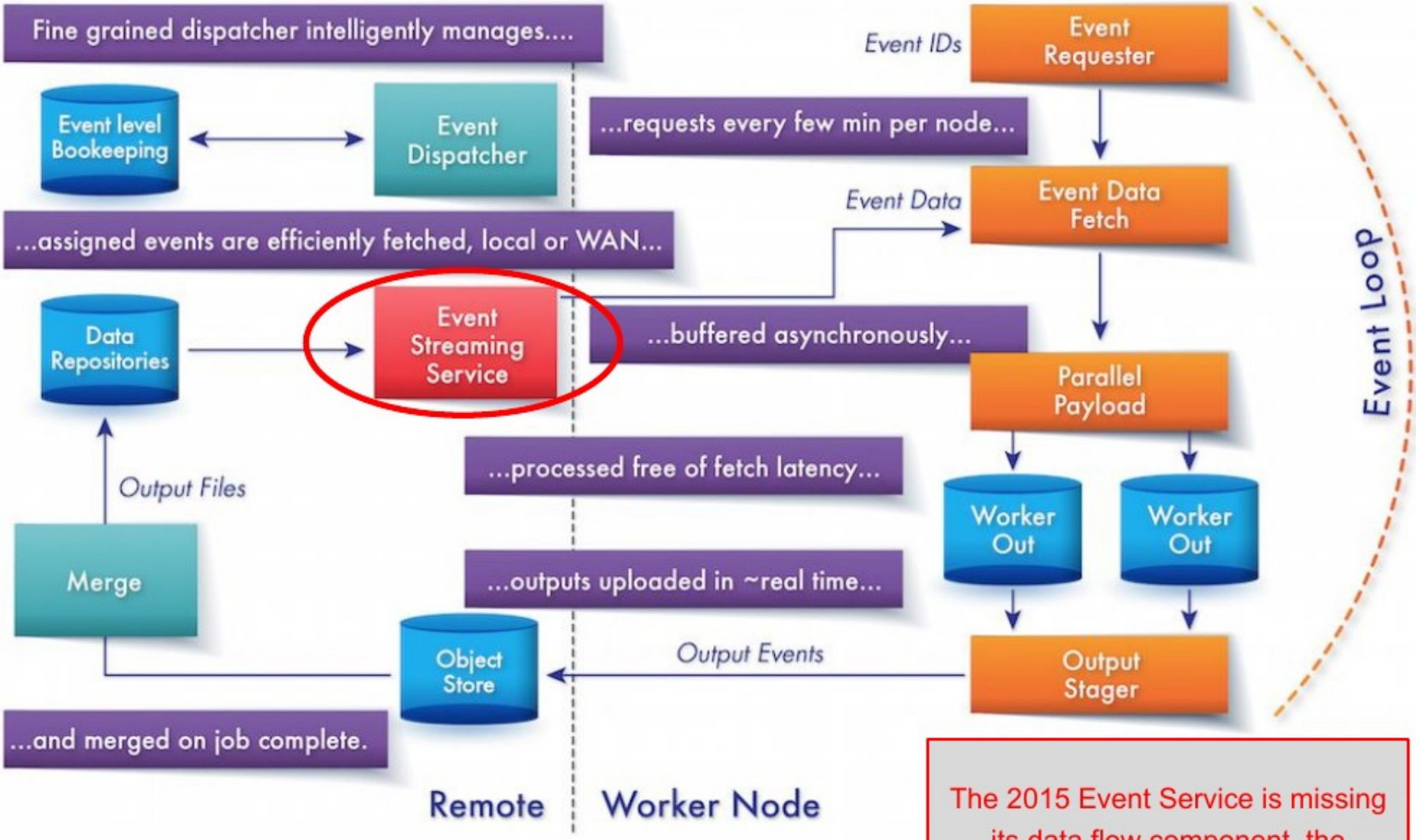
US ATLAS S&C Meeting
Chicago, August 1, 2016

BERKELEY LAB
Lawrence Berkeley National Laboratory

Event Service. Concept

- A fine-grained approach to event processing
  - Quasy-continuous event streaming through worker nodes
- Exploit event processors fully and efficiently through their lifetime
  - Real-time delivery of fine-grained workloads (**Event Ranges**) to running application
  - Be robust against disappearance of the compute resource on short notice
- Decouple processing from chunkiness of files, from data locality considerations and from WAN latency
- Stream outputs away quickly
  - Stage out intermetiade outputs to **Object Stores**
  - Negligible losses if the worker node wanishes
  - Minimal demands for local storage
- Designed for exploiting diverse, distributed, potentially short-lived resources

Event Service in 2015



The 2015 Event Service is missing its data flow component, the Event Streaming Service

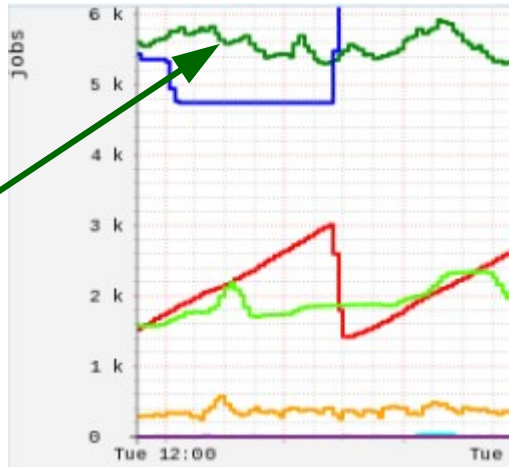
First use-case: ATLAS Geant4 Simulation

- Current implementation of the Event Service can run only ATLAS G4 Simulation
- Simulation uses large fraction of ATLAS CPU-budget on the Grid
  - By offloading Simulation to other platforms (e.g. HPC, clouds) we can free a substantial amount of Grid resources
- G4 simulation is CPU-intensive with minimal I/O requirements
- Meta-data handling is relatively simple
  - Wrt other typed of workloads (e.g. reconstruction)
- ATLAS Geant4 Simulation transform runs in single step

Event Service on various platforms

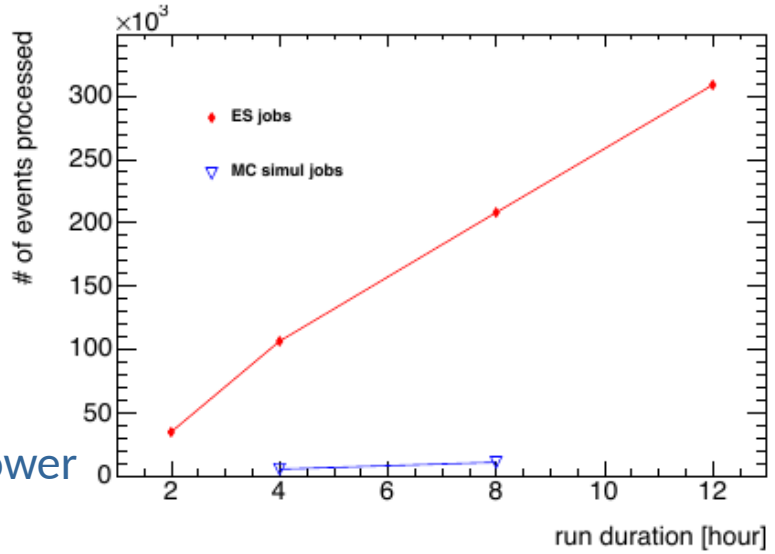
- HPC
  - Special implementation of the Event Service for HPCs is called **Yoda** (a lightweight JEDI)
  - Development and testing platform: **NERSC supercomputers Edison and Cori**
  - **Running production workloads since September 2015**

- Cloud computing
  - Amazon Web Services project at Brookhaven
  - Large-scale Event Service test in late 2015: **"Battle-testing Geant4 10.1"**
  - **Ran steadily at ~5.6K instances with 8-core jobs (~45K cores in total) for more than 24 hours until the run finished**



Event Service on various platforms (contd.)

- P1 HLT Farm
  - Geant4 simulation in Event Service successfully tested at P1 HLT farm in December 2015



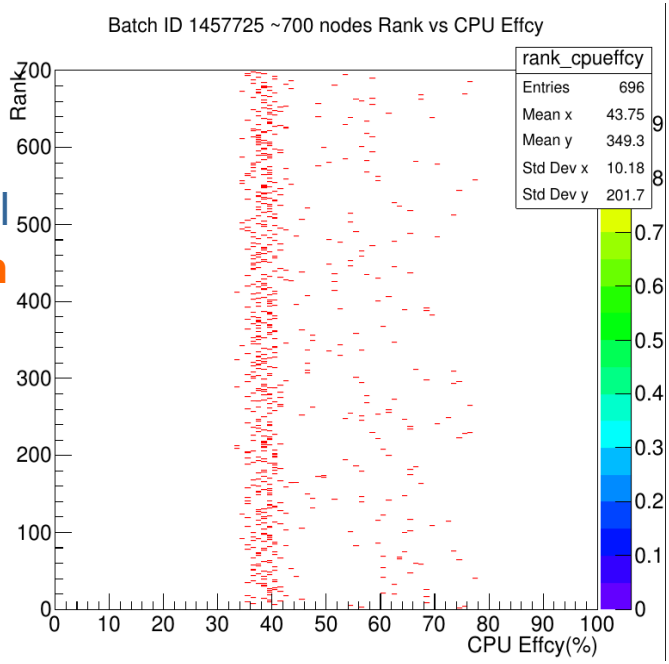
- Volunteer computing
  - Not much progress due to the lack of manpower

- Grid
  - Several Grid sites running Geant4 production workloads in the Event Service (e.g. **CERN, BNL, LRZ-LMU, UKI-NOTHGRID-MAN**)
  - Also, there is a strong interest from Grid sites to use Event Service for **Back-filling**

ES @ HPC. CPU inefficiency

- Yoda has been running ATLAS G4 production workloads on NERSC machines since September 2015. In **2016** our ALCC allocation was charged for more than **13M CPU-hours**

- CPU efficiency of Yoda jobs is not satisfactory
  - **Example**: 700-node job. CPU efficiency of individual compute nodes. More details in **Doug's talk later in this session**



- One known source of the inneficiency:
  - In the present architecture **Yoda must combine multiple PanDA jobs into single MPI-job**
  - Some of PanDA jobs have very few events (the fraction of such jobs is surprisingly high – reason is yet to be understood)
  - If a compute node gets one of such jobs, it finishes event processing early and then stays idle until the end of the entire MPI job

ES @ HPC. CPU inefficiency (contd.)

- Tadashi has proposed a solution to the Yoda innefficiency problem described above

- **Jumbo Jobs**
  - Very large jobs are generated for running at HPC (and other resources)
  - Once the job starts executing on HPC, the pilot retrieves events based on NCore, WallTime and HS06
  - Late binding for events is required (Yoda supports that)

- The concept of Jumbo Jobs is yet to be implemented in practice

- Meanwhile we plan to check if there are other sources of Yoda inefficiency besides the issue with PanDA jobs with small number of events …

Monitoring

- Accounting/monitoring of the Event Service in general and Yoda in particular has been problematic since several months
  - Despite of recent fixes and improvements, there still seem to be some remaining problems, which need to be followed up on

- Yoda sends the following information to PanDA
  - coreCount
  - startTime, endTime
  - ReadEvents and WriteEvents (finished events)
  - Yoda Setup/Running/Stageout Time
  - cpuConsumptionTime
    - **`jobReports.json`** or **`os.times()`** if not killed
    - **`/proc/pid/stat`** if killed

ES/Yoda and the Object Stores

- Event Service/Yoda sends its output files to the Object Stores
  - One output file per event range processed
  - In the case of Simulation 1 Event Range = 1 event. The output files are small: **1-2MB**
- Earlier this year Yoda was staging out files from Edison compute nodes (which have outbound Internet connectivity)
  - Too many small transfers. Difficult to handle by the Object Store (even after recent improvements in the BNL OS)
- Currently output data transfers are completely decoupled from Yoda event processing
  - Yoda stores the outputs to Shared File System
  - Stageout is performed by the Edge Service

Tar/Zip Event Service outputs

- In order to reduce the number of data transfers to Object Stores we implemented a new mechanism in the ES that allows us to tar/zip output files before staging them out

- Pilot tar/zip-s premerged event outputs and sends the tarball to the Object Store
  - Running in Yoda
  - Deployed to the Grid
  - Tests passed on ARC-CE
  - Tests underway on ARC-SuperMUC

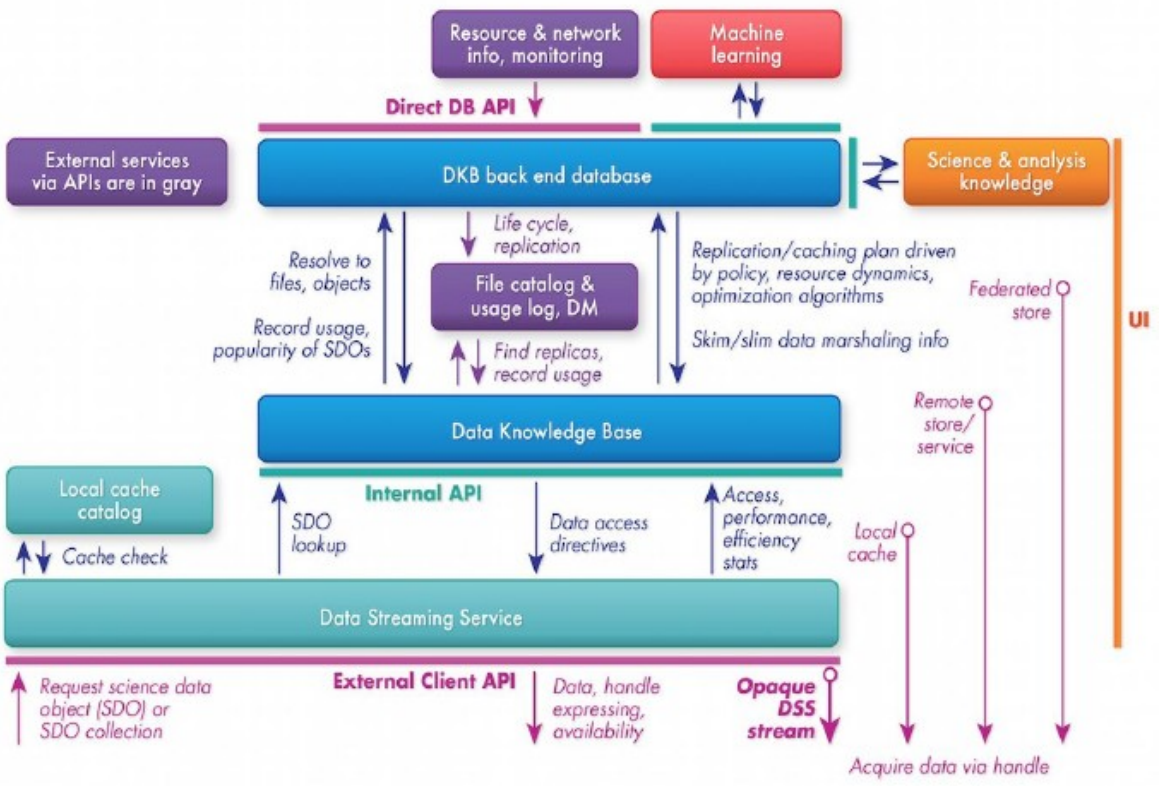
Shared Reader/Writer Processes

- Peter has recently developed a first prototype for the Shared Writer process for AthenaMP simulation
  - By using shared writer process in the Event Service we can produce output files containing more than one event (instead of tar-gzipping single-event files)
  - And by this way reduce the number of output files, that need to be merged later on
- **Shared Reader** process for AthenaMP exists since quite some time, but the testing so far has been limited
- The usage of a Shared Reader process in the Event Service should allow us to implement **asynchronous prefetching** of the input data
  - While Shared Reader is in development/testing, we can try to implement a file-based asynchrounous prefetching mechanism (details under discussion)
- The ability of doing asynchronous prefetching is essential for future development of the **Event Streaming Service (ESS)**

Building the ESS

Two primary components:

- **Data Streaming Service**
  - CDN-like intelligence in finding the most efficient path to the data
  - With minimal replication
  - Data marshaling
  - Smart local caching
- Informed by the **Data Knowledge Base** providing the intelligence on
  - Dynamic resource landscape
  - Science data object (SDO) knowledge
  - Analysis processes & priorities



Backup

Event Deletion

- Rucio first implementation is functional for both logs and the Event Service
- Mapped AGIS Object Store to Rucio RSEs
- A new model is under development for the ES, simpler and more powerful
- PanDA makes create rules and delete rules for the ES outputs (implemented by Tadashi)

Object Store Monitor

- Automated service to test all Object Stores
  - Runs at CERN
  - Runs every 10 minutes
  - Upload and download one 1M file to every gateway host
  - https://rucio-graphite-prod.cern.ch/dashboard/#objectstore

- Object Store issue:
  - Frequently the mean time to upload/download one 1MB file
  - Tar-zipped event file can be 500MB. Timeouts to upload it