Fine grained event processing on HPCs with the ATLAS Yoda system

Paolo Calafiura, Kaushik De, Wen Guan, Tadashi Maeno,
Paul Nilsson, Danila Oleynik, Sergey Panitkin,
Vakho Tsulaia, Peter Van Gemmeren, Torre Wenaus

For the ATLAS Collaboration

CHEP 2015, Okinawa, Japan
April 16, 2015

Motivation

- High Performance Computing (HPC) facilities present unique challenges and opportunities for HEP event processing
- The massive scale of many HPC systems means that fractionally small utilization can yield large returns in processing throughput
- Parallel applications, which can dynamically and efficiently fill any scheduling opportunities the resource presents, benefit both the facility (maximal utilization) and the (compute-limited) science

Challenges

- **Support fine-grained workloads** in order to be able to run efficiently with the variety of scheduling options (from back-filling to large time allocations)
  - ✔ Job granularity changes from files to individual events
  - ✔ Implemented by the **ATLAS Event Service** (see next slide)

- **Leverage MPI mechanisms** for running massively parallel applications on many compute nodes simultaneously
  - ✔ **Yoda** – MPI-based implementation of the Event Service designed specifically for running on HPC systems

- All this has been achieved without changing the existing ATLAS algorithmic code base

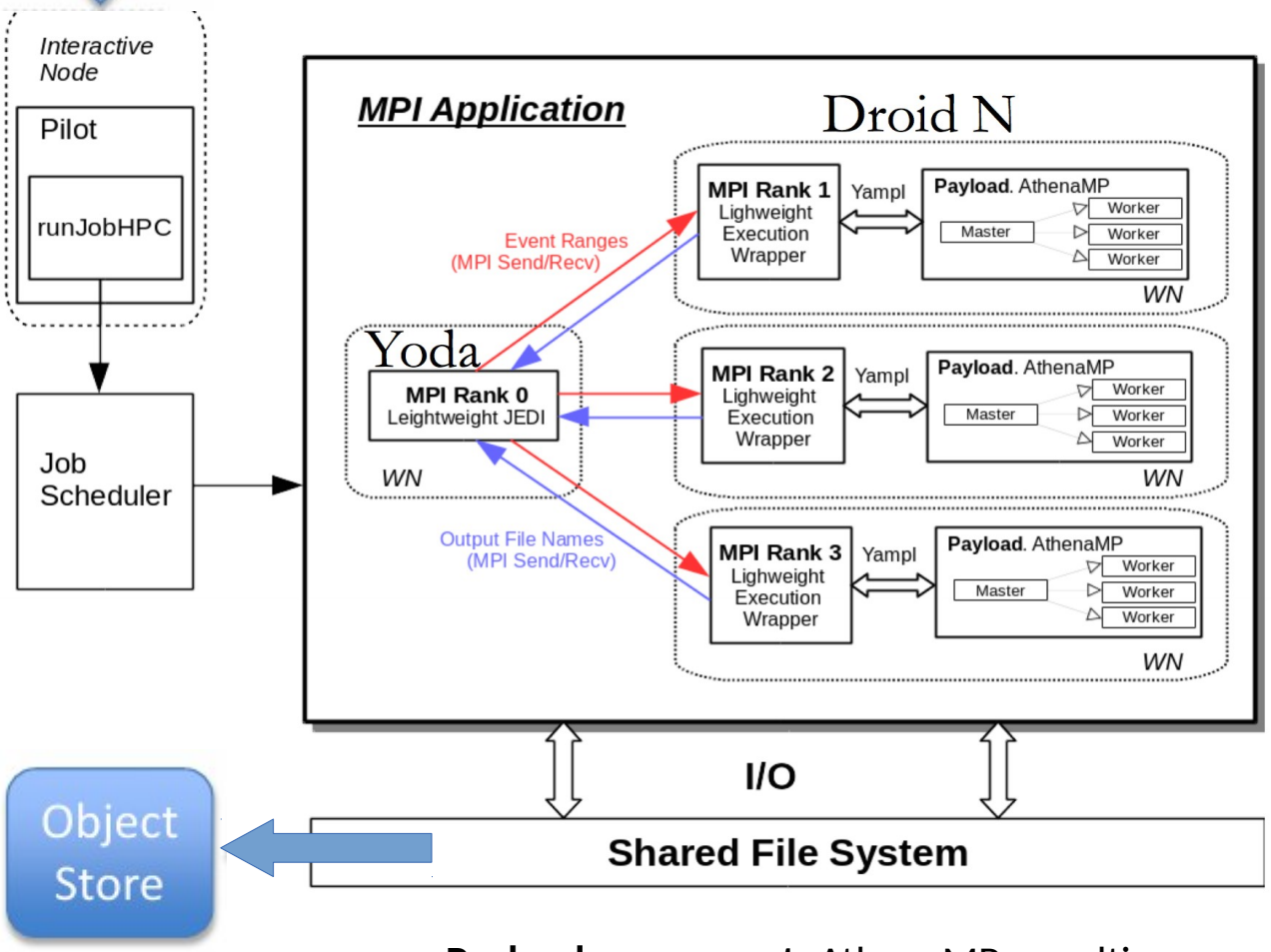
A fine grained Event Service

- **Minimize use of costly storage** in favor of strongly leveraging powerful networks
  - Deliver only those events to a compute node, which will be processed there by the payload application. Don't stage in entire input files
- Event Service is **agile and efficient in exploring diverse, distributed, potentially short-lived** (opportunistic) **resources**
  - 'Conventional resources' (Grid), HPCs, spot market clouds, volunteer computing
- **The job runs either until it uses the entire time slot allocated for it, or until it prematurely gets terminated (resource no longer available)**
  - Minimal data losses
- Applicable to any work-flow that can support fine grained partitioning of the processing and its outputs
- For more details see the presentation by Torre Wenaus at CHEP2015: *"The ATLAS Event Service: A new approach to event processing"* (Contribution #183)

Yoda. Event Service on HPCs

- The 'conventional' Event Service cannot run on most HPC systems
  - Various components of the 'conventional' Event Service communicate to one another over the network using HTTP
  - HPC compute nodes have no internet connection
- Reuse the code of the 'conventional' Event Service wherever possible, …
  - Keep the payload (event processing) component absolutely unchanged
- … just replace HTTP communications with MPI, …
- … and, as a result, we have **Yoda**: a MPI application capable to run within HPC systems with no internet connection
  - Relies on HPC Shared File System for retrieving all necessary input information and for producing outputs

Yoda. Schematic view

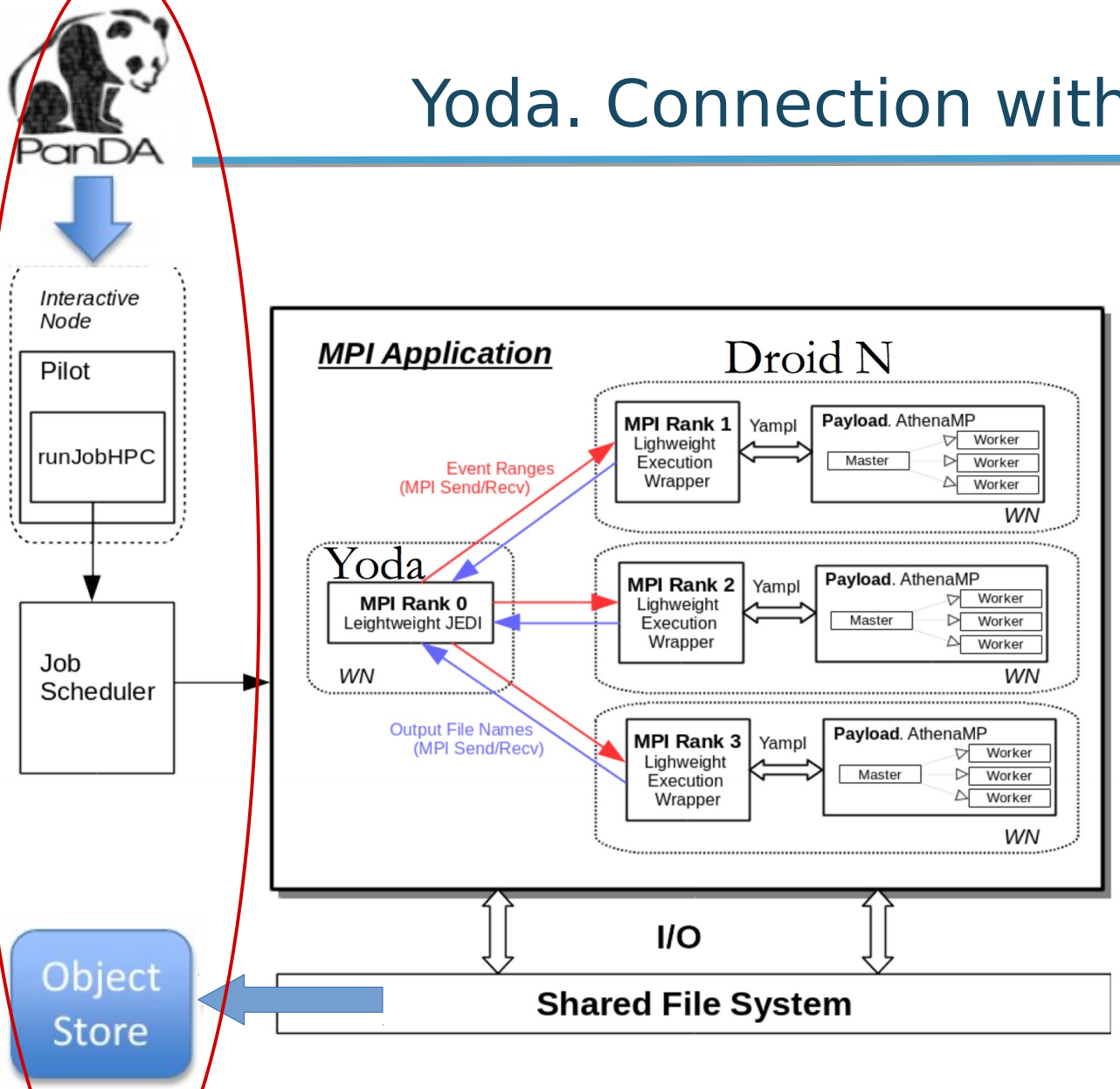


- **MPI application implementing master – slave architecture**
- **Rank 0 (Yoda, master).** Distributes workload between slave ranks
- **Fine grained workload:** individual events or event ranges
- **Rank N (Droid, slave).** Occupies entire compute node;
  Processes assigned workload;
  Saves outputs to the shared file system;
  Asks for the next workload …

- **Payload component:** AthenaMP – multi-process version of the ATLAS simulation, reconstruction and data analysis framework Athena

V.Tsulaia, ATLAS, CHEP2015

Yoda. Connection with PanDA



- **Special implementation of the PanDA Pilot – RunJobHPC – runs on the HPC Interactive Node**
- **Pulls job definitions and input data from the PanDA Server**
- **Submits Yoda jobs to the HPC Batch Scheduler**
- **Streams output files from the Shared File System to the Object Store for final merging**

Job scheduling options

- Yoda is flexible in defining duration and size of MPI jobs
- It offers the efficiency and scheduling flexibility of **preemption** without the application needing to support or utilize checkpointing
  - ✔ AthenaMP payload writes to the disc new output file for each event range
  - ✔ This allows for stopping the job at any time during event processing with minimal losses
- Which means we can run Yoda jobs in the **back-filling mode**
  - ✔ Grab the compute nodes as soon as they become available and use them for the entire duration of their availability
  - ✔ Big 'full' HPCs are full of large hulking rocks; they still have plenty of room for sand for those able to efficiently poor fine-grained work into the cracks

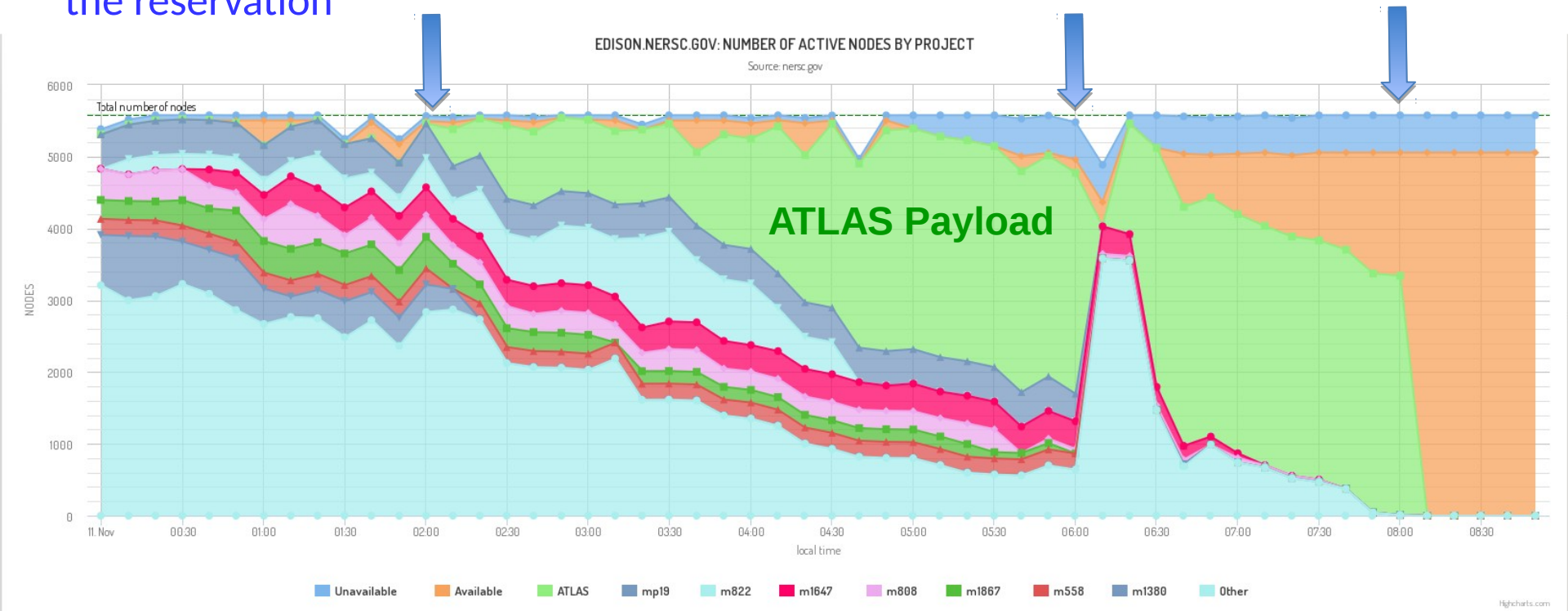
Yoda scavenging resources

- **Edison supercomputer at NERSC (Berkeley, USA)**
- As the machine is emptied either for downtime or for large usage block (**“reservation”**), a **“killable”** queue makes transient cycles available
- Yoda sucks them up and processes the events until the moment they vanish
- … and refills them when they appear again

Edison is getting ready for the reservation

Reservation time

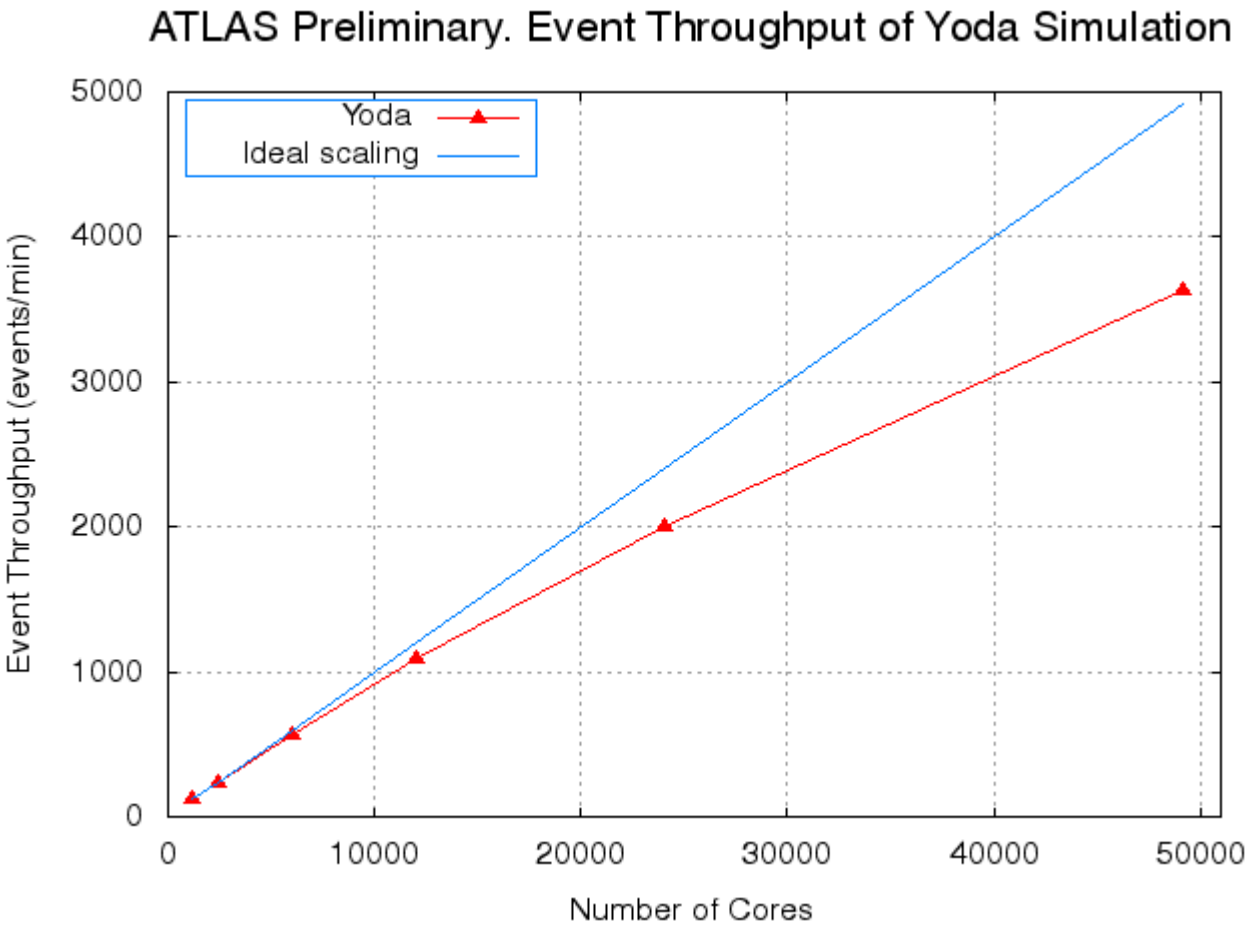
Machine Downtime



Status

- ATLAS Geant4 simulation has been chosen as a first use-case
  - ✔ The biggest return for the least investment
  - ✔ CPU-intensive job with minimal I/O requirements. Meta-data handling relatively simple
- 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 in fall 2014
- In November Yoda was demoed at Supercomputing 2014 as DOE ASCR Demo
- Since then we have been focused on preparation for large-scale productions
- As part of this process we successfully validated for physics the simulation output produced by Yoda jobs

Performance scaling tests



- Recent tests on Edison HPC at NERSC demonstrated that Yoda scales well with the number of parallel processes (cores)
- Although there is still room for improvement
- Simulation of full ATLAS physics events was used for these tests
- **Yoda running with 50K parallel processes simulated 220K events in 1 hour**

Summary

- Yoda – MPI-based Event Service – is our approach to running ATLAS-specific workloads on HPCs
- Thanks to its flexible architecture, Yoda allows for efficient usage of available HPC resources by running the jobs either in large time allocations or in back-filling mode
- Yoda went very rapidly from the concept to the first implementation, which has already been validated for physics, …
- … **and now it is ready for running in production**