Fine grained event processing on HPCs with the ATLAS Yoda system

Paolo Calafiura[1], Kaushik De[2], Wen Guan[3], Tadashi Maeno[4], Paul Nilsson[4], Danila Oleynik[2], Sergey Panitkin[4], Vakhtang Tsulaia[1], Peter Van Gemmeren[5] and Torre Wenaus[4] on behalf of the ATLAS Collaboration

[1]Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA
[2]University of Texas at Arlington, 701 South Nedderman Drive, Arlington, TX 76019, USA
[3]University of Wisconsin, 1150 University Avenue, Madison, WI 53706, USA
[4]Brookhaven National Laboratory, PO Box 5000, Upton, NY 11973, USA
[5]Argonne National Laboratory, 9700 S. Cass Ave, Argonne, IL 60439, USA

E-mail: VTsulaia@lbl.gov

Abstract. High performance computing 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. The ATLAS Yoda system provides this capability to HEP-like event processing applications by implementing event-level processing in an MPI-based master-client model that integrates seamlessly with the more broadly scoped ATLAS Event Service. Fine grained, event level work assignments are intelligently dispatched to parallel workers to sustain full utilization on all cores, with outputs streamed off to destination object stores in near real time with similarly fine granularity, such that processing can proceed until termination with full utilization. The system offers the efficiency and scheduling flexibility of preemption without requiring the application actually support or employ check-pointing. We will present the new Yoda system, its motivations, architecture, implementation, and applications in ATLAS data processing at several US HPC centers.

ATL-SOFT-PROC-2015-004
07 May 2015

1. Introduction

With the increased data volume recorded during LHC Run2 and beyond, it becomes critical for the experiments to not only efficiently use all CPU power available to them, but also to leverage computing resources they don't own. From this perspective, high performance computing resources (HPC) are very valuable for HEP experimental computing. Due to the massive scale of many HPC systems, even fractionally small utilization of their computing power can yield large returns in processing throughput.

Let us consider one example: Edison supercomputer at the National Energy Research Scientific Computing Center (NERSC), Berkeley, USA. With 130K Intel Haswell CPU cores, this machine was #25 on the TOP 500 Supercomputer Sites list in November 2014. One such machine, if hypothetically fully available for the ATLAS experiment [1], could satisfy all ATLAS needs in Geant4 [2] simulation ($\sim$5 billion events/year).

Porting of regular ATLAS workloads (e.g. simulation, reconstruction) to HPC platforms does not come for free. For efficient usage of HPC systems the application needs to be flexible enough to adapt to the variety of scheduling options - from back-filling to large time allocations. In ATLAS this issue has been addressed by implementing a new approach to the event processing, a fine-grained Event Service [3], in which the job granularity changes from input files to individual events or event ranges. After processing each event range, the Event Service saves the output file to a secure location, such that Event Service jobs can be terminated practically at any time with minimal data losses.

Another requirement for efficient running on HPC systems is that the application has to leverage MPI mechanisms in order to be able to run on many compute nodes simultaneously. For this purpose we have developed an MPI-based implementation of the Event Service (Yoda), which is able to run on HPC compute nodes with no internet connectivity with the outside world.

In section 2 of this paper we describe the concept and the architecture of the Event Service. Section 3 describes the implementation details of Yoda and the flexibility it offers in choosing between available job scheduling strategies (back-filling vs allocation). Finally, in section 4 we present the current status of Yoda developments and the results of scaling Yoda up to 50K parallel event processors when running ATLAS Geant4 simulation [4] on the Edison supercomputer at NERSC.

2. ATLAS Event Service

A new implementation of the ATLAS production system [5] includes the JEDI (Job Execution and Definition Interface) extension to PanDA [6], which adds a new functionality to the PanDA server to dynamically break down the tasks based on optimal usage of available processing resources. With this new capability, the tasks can now be broken down at the level of either individual events or event clusters (ranges), as opposed to the traditional file-based task granularity. This allows the recently developed ATLAS Event Service to dynamically deliver to a compute node only that portion of the input data which will be actually processed there by the payload application (simulation, reconstruction, data analysis), thus avoiding costly pre-staging operations for entire data files. The Event Service leverages modern networks for efficient remote data access and highly-scalable object store technologies for data storage. It is agile and efficient in exploring diverse, distributed and potentially short-lived (opportunistic) resources: "conventional resources" (Grid), supercomputers, commercial clouds and volunteer computing.

The Event Service is a complex distributed system in which different components communicate to each other over the network using HTTP. For event processing it uses AthenaMP [7], a process-parallel version of the ATLAS simulation, reconstruction and data analysis framework Athena. A PanDA pilot starts an AthenaMP application on the compute node and waits until it goes through the initialization phase and forks worker processes. After that, the pilot requests an event-based workload from the PanDA JEDI, which is dynamically delivered to the pilot in the form of event ranges. The event range is a string which, together with other information, contains positional numbers of events within the file and an unique file identifier (GUID). The pilot streams event ranges to the running AthenaMP application, which takes care of the event data retrieval, event processing and output file producing (a new output file for each range). The pilot monitors the directory in which the output files are produced, and as they appear sends them to an external aggregation facility (Object Store) for final merging.

3. Yoda - Event Service on HPCs

Supercomputers are one of the important deployment platforms for Event Service applications. However, on most HPC machines there is no internet connection from compute nodes to the outside world. This limitation makes it impossible to run the conventional Event Service on

such systems because the payload component needs to communicate with central services (e.g. job brokerage, data aggregation facilities) over the network.

In Summer 2014 we started to work on an HPC-specific implementation of the Event Service which would leverage MPI for running on multiple compute nodes simultaneously. To speed up the development process and also to preserve all functionality already available in the conventional Event Service, we reused the existing code and implemented lightweight versions of the PanDA JEDI (Yoda, a deminitive Jedi) and the PanDA Pilot (Droid), which communicate to each other over MPI.

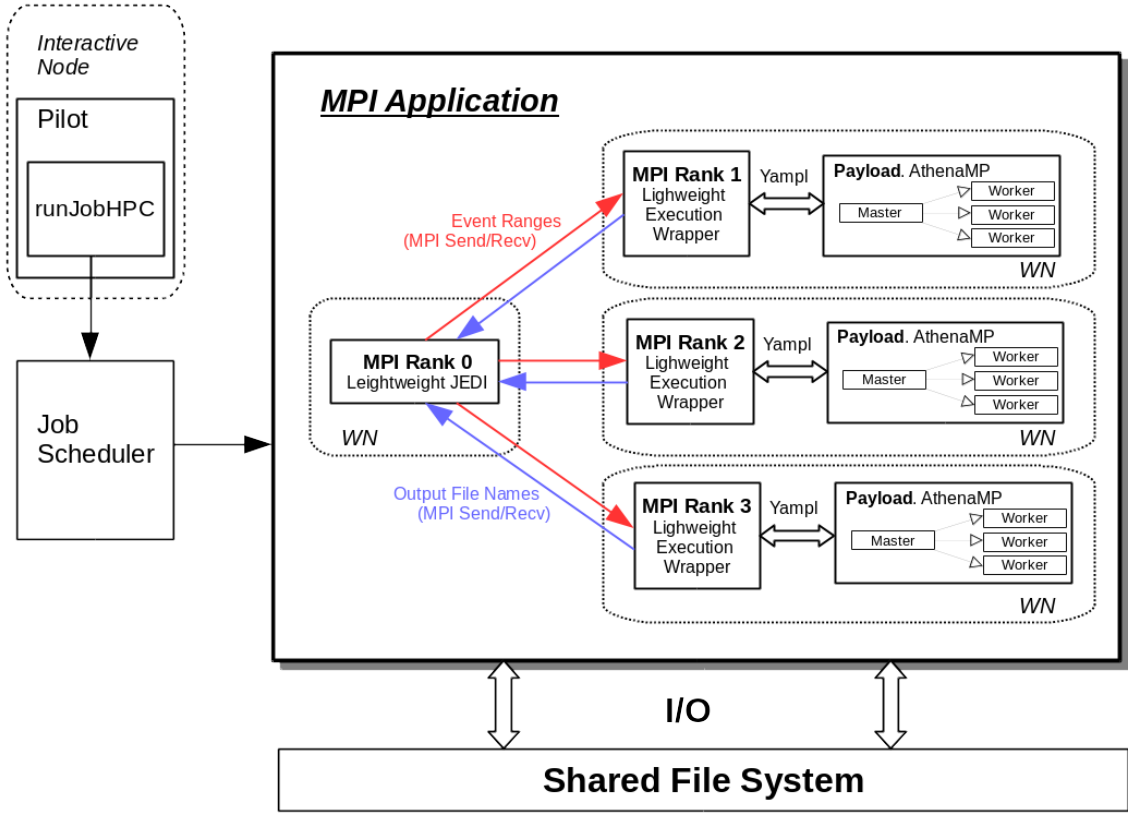

Figure 1. Schematic view of Yoda

Figure 1 shows a schematic of a Yoda application, which implements the master-slave architecture and runs one MPI-rank per compute node. The responsibility of Rank 0 (Yoda, the master) is to send event ranges to other ranks (Droid, the slave) and to collect from them the information about the completed ranges and the produced outputs. Yoda also continuously updates event range statuses in a special table within an SQLite database file on the HPC shared file system. The responsibility of a Droid is to start an AthenaMP payload application on the compute node, to receive event ranges from Yoda, to deliver the ranges to the running payload, to collect information about the completed ranges (e.g. status, output file name and location) and to pass this information back to Yoda.

Yoda distributes event ranges between Droids on a first-come, first-served basis. When some Droid reports completion of an event range, Yoda immediately responds with a new range for

this Droid. In this way, Droids are kept busy until all ranges assigned to the given job have been processed or until the job exceeds its time allocation and gets terminated by the batch scheduler. In the latter case, the data losses caused by such termination are minimal, because the output for each processed event range gets saved immediately in a separate file on the shared file system.

3.1. Connection with PanDA

In order to use Yoda for running ATLAS production workloads, it has to be connected with the ATLAS production system. For this purpose we have developed a special version of the PanDA Pilot (runJobHPC), which provides such connection. The runJobHPC application runs on the interactive compute nodes of HPC systems. Thus, it is able to communicate with central PanDA services over the network. The runJobHPC application pulls job definitions from the PanDA server and stages in all required input files on the HPC shared file system. It submits Yoda jobs to the HPC batch queue, monitors their statuses and also streams out the output files to an external aggregation facility (Object Store), where they are used by separate merge jobs for producing final outputs.

3.2. Job scheduling options

Yoda is flexible in defining duration and size of MPI jobs. We have successfully scaled Geant4 simulation within the Yoda system up to two thousand ranks and have not observed any performance penalties coming from the MPI communication between the ranks. No slowdowns in the average event processing time have been detected either.

On the other hand, the fact that event processors within Yoda write a new output file to the shared file system for each event range gives us the flexibility of preemption without the application need to support or utilize check-pointing. Yoda jobs can be terminated practically at any time with minimal data losses; only the data corresponding to event ranges currently being processed will be lost. This means that Yoda jobs can be submitted to the HPC batch queue in a back-filling mode: the runJobHPC application can detect the availability of a number of HPC compute nodes for certain period of time and promptly submit a properly sized Yoda job to the batch queue for utilizing the available resources.

4. Development status and performance tests

We have chosen ATLAS Geant4 simulation as a first use-case for Yoda (and for the Event Service in general). Simulation jobs used more than half of ATLAS CPU-budget on the Grid in 2014. Thus, by offloading simulation to other computing platforms (e.g. HPC, clouds), we can free a substantial amount of ATLAS Grid resources. Also, simulation is a CPU-intensive application with minimal I/O requirements and relatively simple handling of meta-data. These characteristics allowed us to make rapid progress in the development of a first version of the Event Service, which was delivered in Summer 2014. After that we switched our development efforts to Yoda. By reusing the code of the conventional Event Service, we developed a first working implementation of Yoda in October 2014 and presented it at Supercomputing-2014 in the DOE ASCR demo[1].

In early 2015, Yoda was validated by running a series of ATLAS Geant4 production jobs. The output of these jobs was compared to the output of the same simulation jobs on the Grid. The comparison confirmed that Geant4 simulation within Yoda on HPC and the simulation on the Grid produce the same results.

Also in early 2015 we ran a series of tests on Edison supercomputer with the goal to check how the performance of Yoda scales with the number of MPI-ranks. The results of these tests

[1] http://goo.gl/WSdU4a

are presented on Figure 2, which shows very good scaling of Yoda up to 50K CPU-cores (more than 2K MPI-ranks). The key to such good scaling was to avoid heavy load on the Edison shared file system. This was achieved by delivering ATLAS software releases to the RAM of each compute node. Otherwise the initialization time of Yoda payload applications (AthenaMP) would not scale past 100 MPI-ranks.

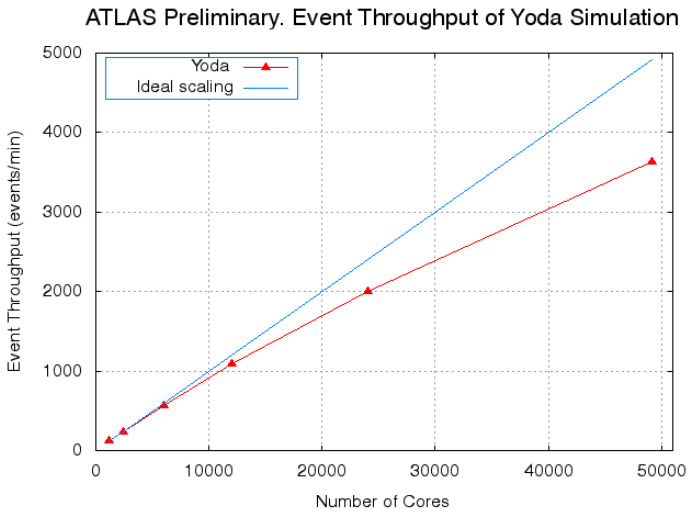

Figure 2. Scaling of the event throughput with number of CPU-cores. ATLAS Geant4 simulation with Yoda on the Edison supercomputer at NERSC

5. Summary

We have developed Yoda, an MPI-based implementation of the Event Service, specifically for running ATLAS workloads on HPCs. ATLAS Geant4 simulations within Yoda have been successfully validated for physics, which proves that Yoda is ready to run ATLAS simulation production workloads on supercomputers. Thanks to its flexible architecture, Yoda allows efficient usage of available HPC resources by running jobs either in large time allocations or in back-filling mode. The performance tests have demonstrated that Yoda scales very well with the number of MPI-ranks, which makes it possible to efficiently run Yoda applications on thousands of HPC compute nodes simultaneously.

6. Acknowledgments

The results presented in this paper have been obtained by using resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] ATLAS Collaboration, 2008 *JINST* **3** S08003
[2] GEANT4 Collaboration, S. Agostinelli et al., 2003 *Nucl. Instrum. Meth. A* **506** 250
[3] Calafiura P et al. on behalf of the ATLAS Collaboration The ATLAS Event Service: A new approach to event processing. Proceedings of the CHEP2015 conference *J. Phys.: Conf. Ser.*
[4] ATLAS Collaboration 2010 ATLAS Simulation Infrastructure *Eur. Phys. J* **C70** 823
[5] De K, Golubkov D, Klimentov A, Potekhin M and Vaniachine A on behalf of the ATLAS Collaboration 2014 Task Management in the New ATLAS Production System *J. Phys.: Conf. Series* **513** 032078
[6] Maeno T for the ATLAS Collaboration 2008 PanDA: Distributed production and distributed analysis system for ATLAS *J. Phys.: Conf. Series* **119** 062036
[7] Binet S et al. 2012 Multicore in production: Advantages and limits of the multiprocess approach in the ATLAS experiment *J. Phys.: Conf. Series* **368** 012018