EPJ Web of Conferences **245**, 05042 (2020) https://doi.org/10.1051/epjconf/202024505042
CHEP 2019

Raythena: a vertically integrated scheduler for ATLAS applications on heterogeneous distributed resources

Miha Muškinja[1,*], *Paolo* Calafiura[1], *Charles* Leggett[1], *Illya* Shapoval[1], and *Vakho* Tsulaia[1] on behalf of the ATLAS Collaboration

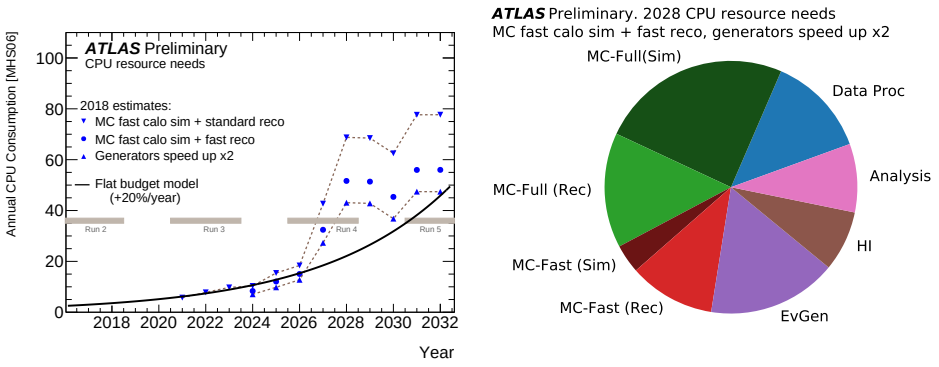
[1]Lawrence Berkeley National Laboratory

Abstract. The ATLAS experiment has successfully integrated High-Performance Computing resources (HPCs) in its production system. Unlike the current generation of HPC systems, and the LHC computing grid, the next generation of supercomputers is expected to be extremely heterogeneous in nature: different systems will have radically different architectures, and most of them will provide partitions optimized for different kinds of workloads. In this work we explore the applicability of concepts and tools realized in Ray (the high-performance distributed execution framework targeting large-scale machine learning applications) to ATLAS event throughput optimization on heterogeneous distributed resources, ranging from traditional grid clusters to Exascale computers. We present a prototype of Raythena, a Ray-based implementation of the ATLAS Event Service (AES), a fine-grained event processing workflow aimed at improving the efficiency of ATLAS workflows on opportunistic resources, specifically HPCs. The AES is implemented as an event processing task farm that distributes packets of events to several worker processes running on multiple nodes. Each worker in the task farm runs an event-processing application (Athena) as a daemon. The whole system is orchestrated by Ray, which assigns work in a distributed, possibly heterogeneous, environment.
For all its flexibility, the AES implementation is currently comprised of multiple separate layers that communicate through ad-hoc command-line and file-based interfaces. The goal of Raythena is to integrate these layers through a feature-rich, efficient application framework. Besides increasing usability and robustness, a vertically integrated scheduler will enable us to explore advanced concepts such as dynamically shaping of workflows to exploit currently available resources, particularly on heterogeneous systems.

1 Introduction

Efficient data processing is of key importance in the ATLAS experiment. For a big majority of its offline data processing the ATLAS experiment is utilizing the Worldwide LHC Computing Grid (WLCG). WLCG is a global collaboration of computer centres used by more than 10 000 physicists that are actively accessing and analysing data in near real-time. During Run 2 data-taking ATLAS collected around 139 fb^{-1} of proton-proton collision events, which required roughly 5 MHS06 of equivalent CPU power and 200 PB of storage for offline-data processing [1]. Going forward, LHC is expected to deliver up to 150 fb^{-1} in Run 3 and

*e-mail: mihamuskinja@lbl.gov



(a) (b)

Figure 1: (a) Estimated CPU resources (in MHS06) needed for the years 2018 to 2032 for both data and simulation processing. The blue points show the improvements possible in three different scenarios, which require significant development work: (1) top curve with fast calo sim used for 75% of the Monte Carlo simulation; (2) middle curve using in addition a faster version of reconstruction; (3) bottom curve, where the time spent in event generation is halved. The solid line shows the amount of resources expected to be available if a flat funding scenario. (b) Fraction of CPU resources needed in 2028 at the end of Run 4 for different processing workflows. The 'MC-Full' section in green is related to the fraction of time spent on the full Atlas Geant4 simulation and divided in a simulation part '(Sim)' for the Geant4 simulation and a reconstruction part '(Rec)' accounting the time spent reconstructing the events. Similarly, the 'MC-Fast' section in red shows this distribution for the time spent running the Fast Calo simulation. Values assume scenario (3)– a faster version of reconstruction and event generation speed-up by a factor of two. Figures taken from ref. [1].

up to 3000 fb^{-1} of proton-proton collision events during the HL-LHC lifetime, which will dramatically increase the CPU consumption and the required disk storage both to analyse the collected data and to generate simulated events with a low enough statistical uncertainty. As shown in figure 1, the CPU consumption is expected to double in Run 3 and increase by a factor of 8 or more in Run 4. Very efficient data-processing frameworks are crucial to handle such large loads and all available computer centres (some not yet built) will need to be utilized for data-processing.

This work explores the possibility of using modern distributed-execution frameworks for ATLAS workflows— specifically Ray [2], which is designed for large-scale machine learning applications and is able to handle thousands of tasks per second across hundred of nodes. The ATLAS experiment has successfully integrated High-Performance Computing resources (HPCs) in its distributed production system. Unlike the current generation of HPC systems, the next generation of supercomputers is expected to be extremely heterogeneous in nature: different systems will have radically different architectures, and most of them will provide partitions optimized for different kinds of workloads (e.g. NERSC's Cori GPU nodes). One of the characteristic differences between HPCs and conventional computing centres is that in HPCs nodes generally do not have outbound network connectivity nor can they be accessed from anywhere apart from few dedicated 'login' nodes in the same HPC system. This is particularly a challenging issue when it comes to integrating the HPCs to the ATLAS distributed production system which usually assumes that each node can access the Panda [3] server (the

ATLAS production and distributed analysis system) to retrieve task information. Further, this limitation means that the data-processing application cannot access any databases over network which can be a requirement for certain data processing steps.

For Run 2 data processing on HPCs the 'Yoda/Droid' workflow [4] was used and only the ATLAS Geant4 simulation tasks were handled. These tasks are suitable for HPCs because they do not require network connection to databases and it is possible to run them with the 'AthenaMP' [5]— the multiprocessing version of the ATLAS data processing framework. AthenaMP is capable of processing multiple events in-flight on a single node and it supports the 'Event Service' (ES) mode where input events are provided on demand by an external application. The advantage of the ES is that the number of input events does not need to be known in advance and that the output is generated on an event-by-event basis. This is suitable for opportunistic resources such as HPCs, where nodes can only be allocated for a certain amount of time (typically few hours) and the most efficient way to use this allocation is to process as many events as possible, i.e. not a pre-determined amount. For example, a typical 'Yoda/Droid' job size at NERSC's Cori HPC for Run 2 ATLAS event simulation used over 100 KNL nodes where a 136-process AthenaMP application ran on each node. ATLAS Genat4 simulation of a single event on a KNL CPU takes from 10 min to 100 min so a central application feeding events to all allocated nodes needs to efficiently schedule more than 20 events per second across more 100 nodes without any bottlenecks.

To continue utilizing HPCs in future and meet the ATLAS CPU needs shown in figure 1, development is needed to scale the job size and efficiently use all available resources that could potentially have largely different architectures in the future. As a continuation of the 'Yoda/Droid' workflow, we developed a Ray-based ATLAS Event Service application. Ray is already designed to perform job orchestration on multiple nodes from a single point with very high frequency and scales well to over a hundred nodes in a cluster. Further advantages of using Ray in place of the current framework is that Ray is widely used by the broader community and centrally maintained, which would eliminate the need of supporting some of the ATLAS home-built software. Further, Ray is easy to install on HPCs (e.g. as a module) which simplifies establishing a single solution for most HPCs. A drawback of Ray is that it requires TCP/IP to perform the distributed scheduling, which might be a limitation for some HPCs (e.g. if they only support MPI communication between nodes). Section 2 lays out the structure of the prototype Ray ES application and shows the performance of realistic-size Geant4 simulation jobs at NERSC's Cori HPC. Cori has two separate systems (partitions): 2388 Haswell nodes and 9668 KNL nodes. Nodes have outbound network connection and inter-node communication is supported with MPI and TCP/IP.

2 Ray-based ATLAS Event Service application

Raythena— the Ray-based ATLAS Event Service application is designed to control a multi-node AthenaMP application from a single node to suit the needs of ATLAS data processing on HPCs. It uses the Ray to connect the allocated nodes in a 'Ray cluster' and perform task scheduling. This connection is achieved via the TCP/IP protocol with a central Redis server running on the Ray head node. Initially, a Ray driver application is launched on the head node. The driver application defines and maintains data and control flows at all times. Further, the driver application triggers construction of the stateful Ray actor applications on all nodes in the cluster. These actor applications are used to control the AthenaMP application– one actor application is constructed on each node (including the head node) and spawns an AthenaMP process via Python's `subprocess.Popen` command. Each actor application

then establishes a communication channel with its own AthenaMP instance via the 'Yampl'[1] protocol and maintains its state. The AthenaMP application goes through initialization and spawns worker processes which process input events. The number of worker processes is configurable and it is usually set to the number of physical CPU cores in a given node. When one of the worker processes on any node is available to process an event, the corresponding actor process requests an input event from the ray driver application and immediately feeds it to the AthenaMP controlling the worker process. A schematic of this workflow is presented in figure 2.

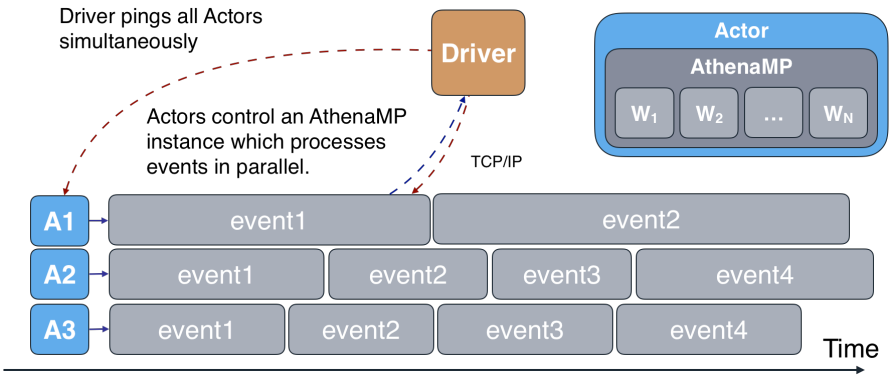


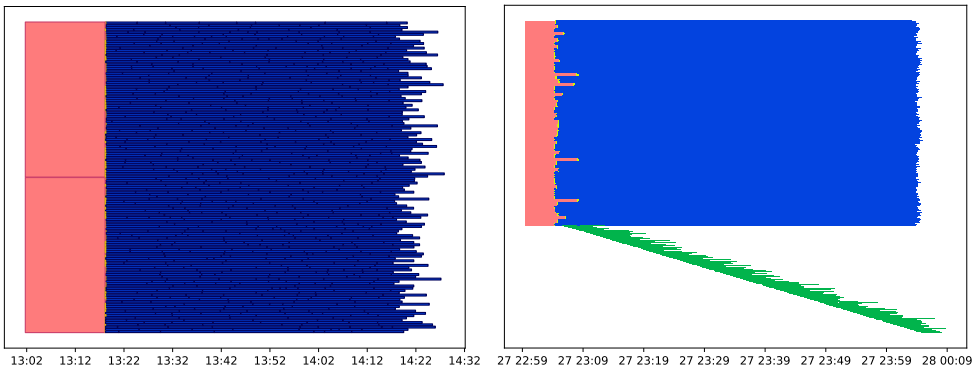
Figure 2: Schematic of Raythena. The application is designed to run on HPCs, where nodes are accessed internally through 'login' nodes. A single driver application (running on any node) controls the 'Ray cluster'. Each node has one Ray Actor application which controls its own instance of an AthenaMP application. The driver application centrally feeds new events to process to available resources and maintains a TCP/IP connection between all nodes in the cluster.

Having a central application that maintains a global state is beneficial for efficient data processing and offers a possibility of error handling and other add-ons in real-time. If an event was unsuccessfully processed, the actor application propagates this information to the driver application. The driver application can then decide, depending on the error type, whether to retry the event processing or not. Further, the Raythena workflow offers an elegant solution for on-the-fly merging of output events. Since the ES workflow produces output on an event-by-event basis, output files need to be merged. We implemented on-the-fly merging in the Ray driver application which is triggered once a pre-defined condition for completed output events is met.

NERSC's Cori HPC system was used to test the performance of the prototype Raythena application. Both Haswell and KNL architectures were tested and example jobs are presented in figure 3. Figure 3a shows an example of a Raythena job running on two KNL nodes where a 64-process AthenaMP instance ran on both nodes. ATLAS Geant4 Inner Detector only simulation tasks were ran to increase the turnaround time and potentially increase the possibility of a bottleneck in scheduling. It is evident that there is no downtime between event processing compared to the time it takes to process each event, which indicates that there are no bottlenecks in the Ray application. Further, figure 3b shows an example of a 60-node job on Cori Haswell with 32-process AthenaMP applications. Additionally, on-the-fly merging was triggered for every 100 processed events. Merged jobs are separate Athena applications

[1] Yampl is a lightweight and robust IPC library developed in ATLAS for inter-process communication

and visible in green rectangles in the bottom part of the figure. Since all cores were utilized by the AthenaMP application, merge jobs were done with oversubscribing. No bottlenecks are seen in the 60-node example which gives us good confidence that Ray is capable of running ATLAS workflows on HPCs.



(a) (b)

Figure 3: CPU utilization at NERSC's Cori HPC with the Ray-based ATLAS Event Service prototype application. ATLAS Geant4 Inner Detector only simulation tasks were ran. Blue rectangles represent single events being processed– from start time to finish. Red rectangles are Athena initialization times and green rectangles between red and blue rectangles are initialization times of individual AthenaMP worker processes. (a) Example of a two-node application running on Cori NKL with a 64-process application on each node. (b) Example of a 60-node application running on Cori Haswell with a 32-process application on each node. Additionally, Athena 'merge' jobs were spawned for every 100 processed events, which are visible in green rectangles in the bottom part of the figure.

There is ongoing development to turn the Raythena prototype application in a production-quality Event Service application that would be used as the default job orchestrating workflow on HPCs in Run 3 instead of the 'Yoda/Droid' application. The stand-alone Raythena version presented in this work needs to be extended so that it can be automatically launched once workload is available. This will be achieved by connecting it with the Harvester application [6] which retrieves task information from the Panda server. Further, the Pilot2 [7] application will be used to handle AthenaMP process instead of the stand-alone solution presented in this work. The advantage of using Pilot2 instead of a direct link with AthenaMP is that Pilot2 is already an established application that also performs monitoring, logging, and error handling. Further, Pilot2 will also be used as a middleware application on regular computing centres which will make HPC tasks more similar to the regular tasks and make the maintenance easier.

3 Summary

The increased luminosity in LHC's Run 3 and Run 4 demands development of efficient data-processing systems to keep up with the ever increasing amount of data the LHC experiments are generating. In this work, we presented a Ray-based prototype of the ATLAS Event Service (Raythena) that was successfully tested running an ATLAS Geant4 simulation on 60

nodes in parallel. Raythena is based on the distributed execution framework Ray and offers a scalable solution for the ATLAS Event Service workflow that can be used on HPCs. The ongoing development aims to deliver a job orchestrating workflow based on Raythena that will be used as the default on all HPCs.

Ray has proven to be a versatile tool suitable for HEP distributed computing and with its rich features it could be exploited even further. One potential application would be to reproduce entire analysis workflows (e.g. from event generation to statistical analysis) in a massively parallel manner on HPCs with a very quick turn-around. A benefit of such an application would be that it could have feedback loops (which Ray excels at) where tasks would dynamically change based on already obtained results and enable very fine-grained calculations or optimizations.

References

[1] The ATLAS Collaboration, *Computing and software public results*, `https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults`

[2] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M.I. Jordan et al., *Ray: A Distributed Framework for Emerging AI Applications*, in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (USENIX Association, Carlsbad, CA, 2018), pp. 561–577, ISBN 978-1-939133-08-3, `https://www.usenix.org/conference/osdi18/presentation/moritz`

[3] P. Nilsson, M. Potekhin, T. Maeno, J. Caballero, K. De, T. Wenaus, PoS p. 027 (2008)

[4] P. Calafiura, K. De, W. Guan, T. Maeno, P. Nilsson, D. Oleynik, S. Panitkin, V. Tsulaia, P.V. Gemmeren, T. Wenaus, Journal of Physics: Conference Series **664**, 092025 (2015)

[5] P. Calafiura, C. Leggett, R. Seuster, V. Tsulaia, P. Van Gemmeren, J. Phys. Conf. Ser. **664**, 072050 (2015)

[6] T. Maeno, F.H. Barreiro Megino, D. Benjamin, D. Cameron, J.T. Childers, K. De, A. De Salvo, A. Filipcic, J. Hover, F. Lin et al. (ATLAS Collaboration) (2018)

[7] P. Nilsson, A. Anisenkov, D. Benjamin, D. Drizhuk, W. Guan, M. Lassnig, D. Oleynik, P. Svirin, T. Wegner (ATLAS), EPJ Web Conf. **214**, 03054 (2019)

Acknowledgement

This work was supported in part by the U.S. Department of Energy, Office of Science - High Energy Physics, and it used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.