Computing and Informatics, Vol. XX, 2021, 1–14, V 2021-Oct-8

THE TRIGGER PERFORMANCE MONITORING AND RATE PREDICTIONS PREPARATION FOR RUN 3 AT ATLAS EXPERIMENT

Aleksandra Poreba on behalf of the ATLAS Collaboration

CERN
211 Geneva 23
Switzerland
e-mail: aleksandra.poreba@cern.ch

Abstract.

Bespoke Cost Monitoring software collates data on the performance of all aspects of the ATLAS experiment's High Level Trigger software. These data are exported for subsequent analysis offline, and are used to understand the resource usage of the individual trigger selections in terms of the amount of CPU time and the amount of raw detector data which was required to perform the selection.

For the LHC's Run 3, the ATLAS High Level Trigger is re-implemented in a multi-threaded framework with both intra-event and inter-event algorithm parallelism. We will describe some of the complications and considerations which arise from monitoring event metrics in a highly parallel environment.

Keywords: CERN, ATLAS, Trigger, Cost Monitoring

Mathematics Subject Classification 2010: AB-XYZ

1 INTRODUCTION

The ATLAS experiment [1] at the LHC is a multipurpose particle detector designed to record physics events for a wide range of measurements and searches. It consists of inner and muon tracking detectors, electromagnetic and hadronic calorimeters, and uses magnetic fields produced by solenoid and toroid magnets. The detector

records the collisions delivered by the LHC at a rate of 40 MHz (for proton-proton data taking).

The ATLAS experiment was not designed to record all these collisions, as most of these are not interesting for physics analysis. The ATLAS trigger system (architecture is presented in the Figure 1) aims to select the interesting collisions in real-time. The first-level trigger is implemented in hardware and uses a subset of the detector information to accept events at a rate below 100 kHz. It is followed by a software-based trigger HLT (High Level Trigger) that reduces the accepted event rate to 1.5 kHz on average, depending on the data-taking conditions. An extensive software suite [2] is used for real and simulated data reconstruction and analysis, for operation and in the trigger and data acquisition systems of the experiment.

The HLT runs on a large computing farm of CPUs. During the shut-down between data-taking periods, the machines are being upgraded in order to achieve higher processing efficiency. However, estimations of HLT resources usage are also required to inspect if the new software will be able to process the data with a desired frequency. The Trigger system is equipped with such tools to predict the software performance in advance of collisions.

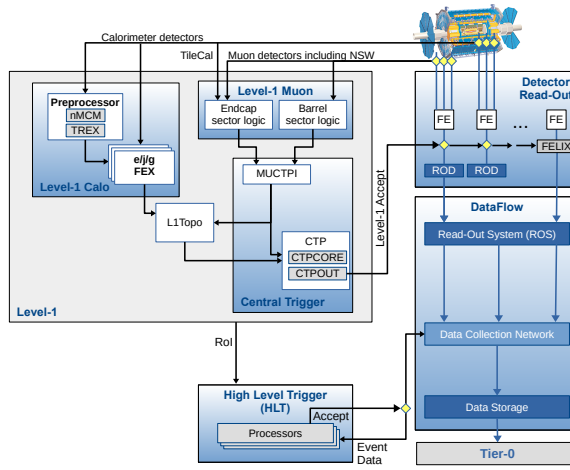


Fig. 1: Functional diagram of the ATLAS Trigger and Data Acquisition system in Run 3 [3].

1.1 Trigger Menu

The HLT filtering process consists of selection steps, referred to as chains. Each chain has defined physics thresholds that are required to consider an event as interesting

for physics analysis. The chains are defined in the Trigger Menu, different menus are prepared for different data taking types (heavy ion, proton-proton physics).

In each step, a series of physics reconstruction algorithms are executed in order to retrieve the details of the recorded event. Afterwards, the chains are tested to check if the given chain's criteria were fulfilled. If not, further algorithms won't be executed and an event is rejected. The chain's steps are processed until either the chain rejects the event at the end of a step, or passes the final step and hence passes the event.

The order of algorithms within a chain relies on an early-rejection mechanism. The early steps should be the most selective and rejective. If a step fails, steps that depend on it are removed and, as a consequence, many chains will be rejected. This way the computing resources usage can be reduced.

1.2 Cost Monitoring

During online data taking or data reprocessing monitoring data is collected in addition to physics data. One of the available monitors is the Cost Monitor that enables data-driven CPU usage predictions. The "cost" stands for the execution time of algorithms or events, based on which the resources estimations can be performed in advance of collisions. In the Trigger development phase, it can be used to spot problematic resource usage and optimize the framework.

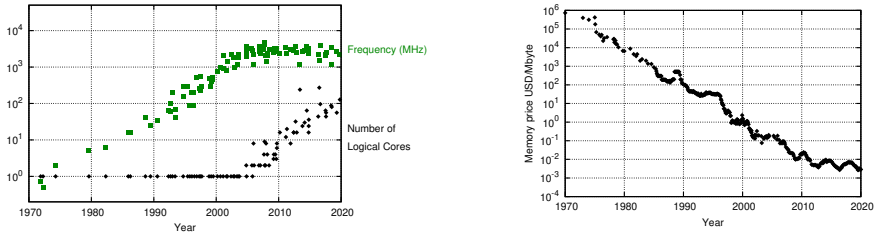
When collecting the data, 10% of all the events are monitored by the Cost Monitor. With the HLT execution reaching up to 100 kHz rate, up to 10 kHz of monitoring data is recorded. Statistics relating to the execution of the algorithm and its request for raw detector data are saved to perform the offline analysis: start and stop timestamps, requested data size, and status flags.

The Cost processing is based on a separate HLT output data stream containing recorded cost monitoring details. After processing the monitoring data a ROOT [4] file is produced with a set of histograms illustrating the execution performance of different Trigger areas, for example, algorithms, chains, threads occupancy, or the read out system. A set of Comma Separated Value files is created from the monitoring histograms with different available summaries presenting alternate overviews of the resource usage. The result can be displayed on the Trigger Cost Browser website, equipped with tools to enable easy browsing of the summary tables. The process and result structure was described in detail in Ref. [5].

1.3 Multithreaded Framework

The frameworks used in the Trigger, Athena [6] and Gaudi [7], were designed for single threaded operations. Since that time the computing market has changed to favoring multi-core processors for higher processing efficiency. The same is not true for memory; the prices are oscillating over the last years and no longer matching the decreasing trend. The observed trends are presented in Figures 2a and 2b. These

changes encourage a concurrent framework approach with shared memory rather than single threaded processing.



(a) (b)

Fig. 2: Left: CPU frequency and logical core count trends are presented based on data [8]. Right: illustrates memory price trends between 1975 and 2019, based on data [9].

During Run 1 and 2 chains and their algorithms were executed linearly using a single-threaded approach. The results of algorithm executions were cached and could be reused in the future if required by other chains to minimize the computing costs.

For Run 2 the framework was upgraded to support a multi-processing approach with Athena instances being forked after initialization. The processes shared the memory at the OS level as long as the pages stayed the same. In case of any change in execution between forks, each one received its own copy of that page.

For Run 3 both Athena and Gaudi are redesigned to support both the multi-process and multi-threaded mode. The new version of Athena, AthenaMT [10] is based on Gaudi Hive [11] with a concurrent task scheduler based on Intel Thread Building Blocks (TBB) [12].

The High Level Trigger was re-implemented in the new concurrent framework. Instead of processing a single chain at a time then caching a result, a step is now executed for all of the chains with the same input only once. The processing is parallel both inter-event (multiple events can be processed in parallel), intra-event (algorithms for one event can be executed in parallel provided they have needed input), and also in-algorithm (algorithms can use multithreading specific operations, for example concurrent loops). A schematic example of concurrent events processing in HLT is shown in Figure 3.

2 COST MONITORING IN ATHENAMT

The multi-threading redesign changes also affected the Cost Monitoring. An algorithm execution can no longer be associated with only one chain, which required revising the way the Cost data is created.

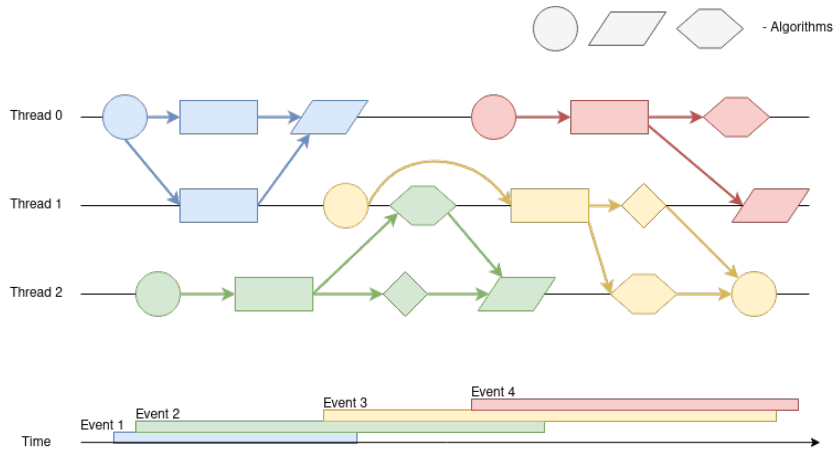


Fig. 3: Schematic flowchart describing concurrent event processing in AthenaMT. Each shape corresponds to a type of algorithm and each colour corresponds to data associated with one event. Data flow is represented by arrows.

Which chains actually executed the algorithm (i.e. chains which weren't already rejected in an earlier step) can be determined based on the navigation data generated during the HLT's execution. After each algorithm execution the summary with a result is created and, in the end, saved in the output data stream file for further physics analysis. Based on these results the Cost "Chain Summary" is created by checking if a chain's algorithms navigation item was created for the processed event.

In AthenaMT the execution time is the same for all chains that include a given algorithm. In the past if the algorithm result was cached, the second execution time was 0 - cached result was used. This created an illusion of "free" chains, that used only cached results when in fact the cost was spread between other chains that were executed earlier. The new approach is more reliable showing the true cost of the chain.

Table 1 presents a comparison of Run 2 and Run 3 cost performance results for two chains: selections A + B and A + C in the order of execution. In the Run 2 Cost Summary the execution time of the A + C chain (executed later than the A + B chain) is only 50 ms - cached A result from the first executed chain was used and only C selection result had to be obtained. In AthenaMT the real execution time of a chain is shown. In the A + C chain the A execution is no longer hidden as a result of caching and the chain execution time does not depend on the order of the chains.

If the order was switched, the results for Run 2 would be different: 250 ms for A + C chain and 100 ms for the A + B chain. The new Cost Monitoring framework is resistant to such changes.

Algorithm	Execution Time [ms]
Selection A	200
Selection B	100
Selection C	50

Chain	Execution time per event in Run 2 [ms]	Execution time per event in Run 3 [ms]
Selections A + B	300	300
Selections A + C	50	250
Total measured time	350	350

Table 1: Representative numbers of mean execution time of two chains containing selections: A+B and A+C recorded by the Cost Monitoring in Run 2 and Run 3 framework.

On the other hand, with the new way of measuring chains' execution time, the sum can be higher than the time of execution of the event. Knowing that two chains can execute one algorithm, the sum of all the chains' execution times won't be equal to the total time of algorithms' execution. The new changes make a chain's time more accurate when comparing to each other. For a global HLT cost analysis, the time and number of calls for each algorithm are the most important.

2.1 Saving Cost Monitoring Data

The new framework defines a number of processing slots. Incoming events are assigned to an available slot and the event will occupy the slot until it has finished processing. Each slot has its own data store, where the details of the monitored event are saved. Cost monitoring data from all of the slots are saved to the same output data stream file.

However, during online data taking and reprocessing, additional details can be saved to the Master Slot (set in configuration, by default Slot 0). When the event processed by this slot is monitored by cost, data about events from other slots is collected also into the master slot's data store. The data saving mechanism is illustrated on Figure 4.

This mechanism is used for thread monitoring in order to retrieve details of what is happening in other slots during current event processing. Events are processed at the same time on different slots (mentioned in the section 1.3 intra event parallelism). With the full information we are able to compare the threading performance of the HLT process over all slots and assigned threads.

The information about the source slot is also saved to the output data stream, necessary for the Cost Processing. Events originated in non-master slots that are saved in master slot's data store should be ignored for other monitors, for example

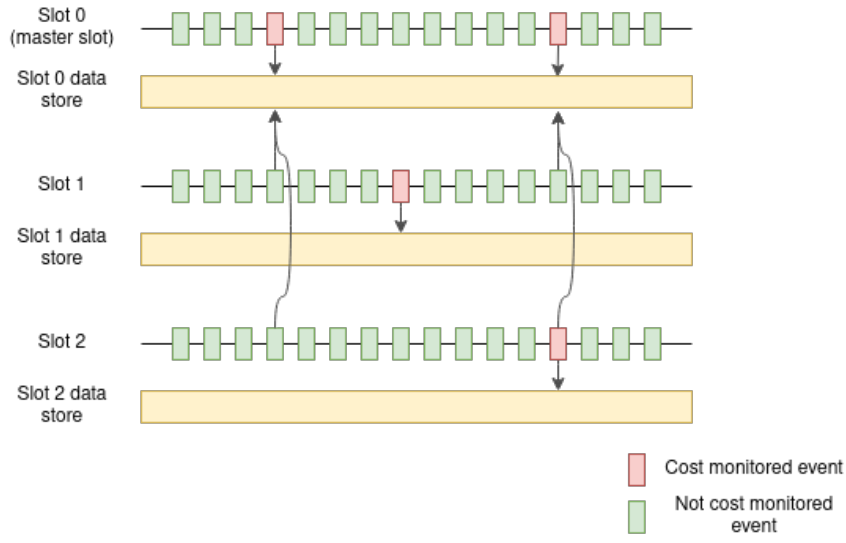


Fig. 4: Diagram illustrating cost monitoring data saving mechanism with additional data saved to the master slot. Three cases are possible: master slot is monitored, non-master slot is monitored or both master and non-master slots are monitored.

for algorithms and chains. Otherwise, the same data would be processed twice (the third case on the Figure 4) and the monitored event rate would be higher than 1 in 10. In order to avoid the data bias, the thread monitoring uses only the master slot saved data.

2.2 Thread Monitoring

Thread monitoring is a new feature possible inside a multi-threaded HLT implementation. Its purpose is to give an overview of the multi-threading efficiency for given forks, event slots and threads configuration. The event execution time is not only determined by all the algorithms execution time but also for "framework time". It is defined as the time a thread spent outside of scheduled algorithms while waiting for an algorithm to be dispatched.

By looking at how much time it took to process an event (from the earliest algorithm executed on any of the threads for this event to the latest) and the sum of all the algorithm's execution on each individual thread, the time spent on executing algorithms and on framework related operations can be determined. Algorithms from other slots whose execution spans the global start and stop of the master thread add a degree of uncertainty to this measurement, this component is computed separately as unmonitored time. The three values: algorithm, framework and unmonitored time are illustrated on Figure 5. The way they are computed is described by equations 1, 2 and 3.

$$\text{eventTime} = \text{globalStart} - \text{globalEnd} \tag{1}$$

$$\text{unmonitoredTime} = (\text{globalStart} - \text{threadStart}) + (\text{threadEnd} - \text{globalEnd}) \tag{2}$$

$$\text{frameworkTime} = \text{eventTime} - \text{unmonitoredTime} - \text{algorithmTime} \tag{3}$$

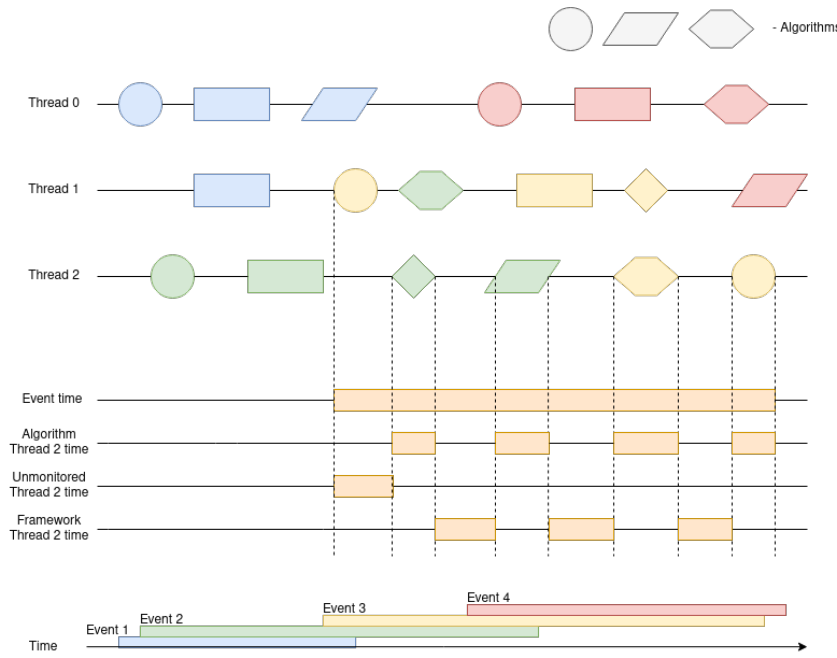
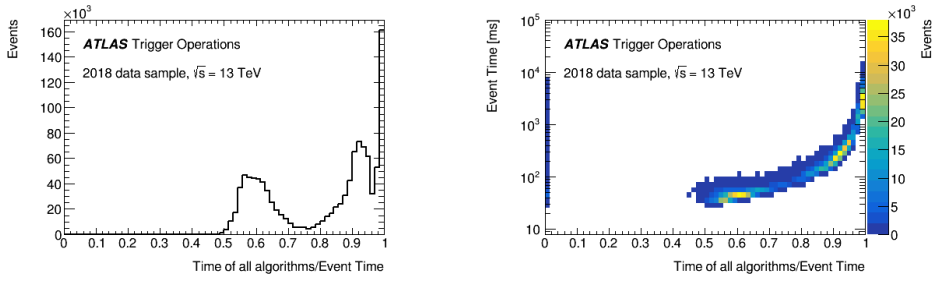


Fig. 5: Graphic interpretation of event, unmonitored and framework time for Thread 2 based on the flowchart 3. During the event's time window all algorithms are monitored (not only algorithms that are running for given event).

Figures 6 and 7 show the fraction of algorithm and framework time. It is observed that in long events, algorithm time dominates whereas in short events the fraction is closer to being equal, testifying to the good performance of the early rejection system. The framework time is consistent in all events. Different time of processing an event depends on the actual event details. The algorithms are only executed when the conditions are fulfilled and the chain wasn't rejected earlier. The mean framework time should stay the same for a given configuration, only the algorithm time varies. Analysing the distribution of algorithm time per event and framework time per event the best configuration can be identified.



(a) (b)

Fig. 6: Example of measured time of all the algorithms executed on the thread as fraction of the monitored event time window. On the right plot the fractional time is showed as a function of the monitored event time window. Three peaks can be observed in the plots - one when thread processes algorithms 60% of total time, which happens for short events (approximately 100 ms), when the event data doesn't fulfill the requirements to trigger execution of time consuming algorithms. The other two peaks represent long events (approximately 300 ms and 3000 ms) when algorithm processing takes majority of the total time. The peaks correlate with recorded times of algorithm execution per event. For some of the events included in the data sample the algorithm processing takes 0% of the time, which is happening when the event does not fulfill requirements to trigger any of the algorithm executions. The rest of the time is spent for so-called "framework time". The plots were created with a small 2018 data sample that was preloaded into the HLT farm and processed repeatedly. [13]

3 INTERPRETING THE COST RESULTS

Cost monitoring was written into the trigger infrastructure in order to perform detailed offline studies of the system's performance. Based on Cost data collected during the data taking, one can not only estimate CPU resources but also access the details of the HLT execution. The Cost summaries cover executed algorithms, chains, sequences (groups of algorithms), data requests and the aforementioned thread occupancies. One additional summary covers the global parameters of the HLT execution, including mean processing time of the event, number of active events within a given period of time, or number of algorithm calls in the event.

In the following paragraphs the contents of some of the summaries will be discussed. Starting with the "Algorithm Summary", different algorithms and their parameters can be analysed, including:

- Number of calls per event,
- Mean time of execution per event,
- Total algorithm execution time and what fraction of total HLT walltime it is,

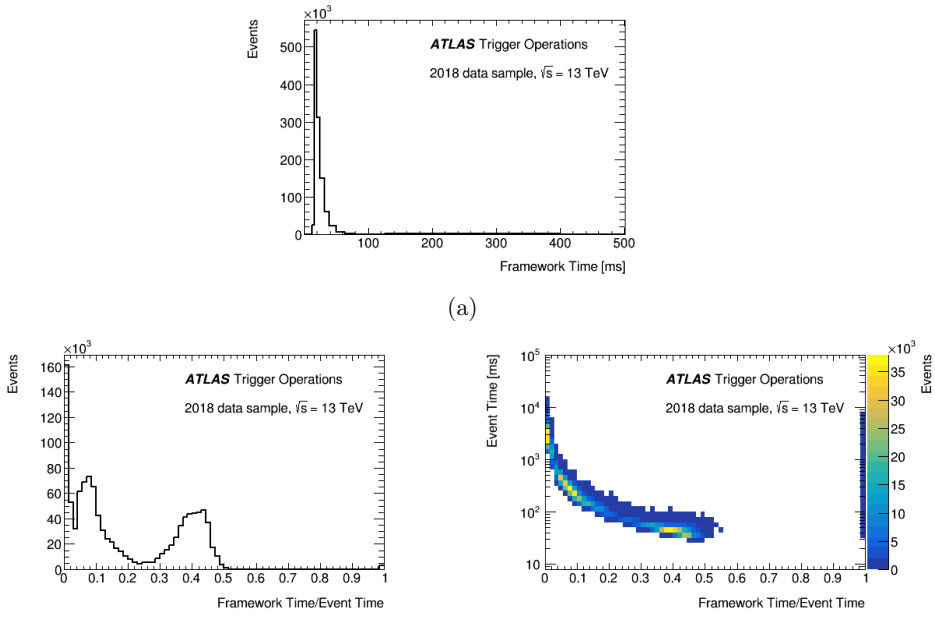
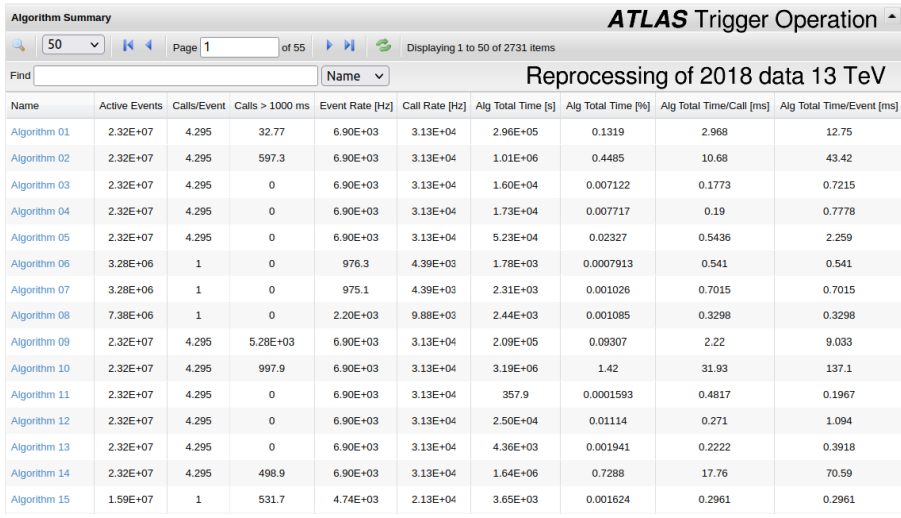


Fig. 7: Example of measured framework time on one thread as fraction of the monitored event time window. On the plot 7c the fractional time is showed as a function of the monitored event time window. Three peaks can be observed - one when thread performs framework operations 40% of the time during short events processing (approximately 100 ms). The other two correspond to long events (approximately 300 ms and 3000 ms) when the framework time takes a minor fraction of the event time window. For some of the events included in the data sample the framework time takes 100%, what is happening when the event does not fulfill requirements to trigger any of the algorithm executions. Based on the plot 7a it can be observed that for all events the framework time is stable and its mean value equals to 20 ms, despite fluctuations in framework fraction time. The plots were created with a small 2018 data sample that was preloaded into the HLT farm and processed repeatedly. [13]

- How often the algorithm was called and how often algorithm was executed in an event,
- If the algorithm requested any data and if these data were cached locally or fetched over the network.

An example of "Algorithm Summary" table is presented in Figure 8.

The expensive algorithms can be identified by a long algorithm total time. Not only the measured time is important but also what percentage of the total time the execution took.



Algorithm Summary — **ATLAS** Trigger Operation

50 | Page 1 of 55 | Displaying 1 to 50 of 2731 items

Find | Name | Reprocessing of 2018 data 13 TeV

Name	Active Events	Calls/Event	Calls > 1000 ms	Event Rate [Hz]	Call Rate [Hz]	Alg Total Time [s]	Alg Total Time [%]	Alg Total Time/Call [ms]	Alg Total Time/Event [ms]
Algorithm 01	2.32E+07	4.295	32.77	6.90E+03	3.13E+04	2.96E+05	0.1319	2.968	12.75
Algorithm 02	2.32E+07	4.295	597.3	6.90E+03	3.13E+04	1.01E+06	0.4485	10.68	43.42
Algorithm 03	2.32E+07	4.295	0	6.90E+03	3.13E+04	1.60E+04	0.007122	0.1773	0.7215
Algorithm 04	2.32E+07	4.295	0	6.90E+03	3.13E+04	1.73E+04	0.007717	0.19	0.7778
Algorithm 05	2.32E+07	4.295	0	6.90E+03	3.13E+04	5.23E+04	0.02327	0.5436	2.259
Algorithm 06	3.28E+06	1	0	976.3	4.39E+03	1.78E+03	0.0007913	0.541	0.541
Algorithm 07	3.28E+06	1	0	975.1	4.39E+03	2.31E+03	0.001026	0.7015	0.7015
Algorithm 08	7.38E+06	1	0	2.20E+03	9.88E+03	2.44E+03	0.001085	0.3298	0.3298
Algorithm 09	2.32E+07	4.295	5.28E+03	6.90E+03	3.13E+04	2.09E+05	0.09307	2.22	9.033
Algorithm 10	2.32E+07	4.295	997.9	6.90E+03	3.13E+04	3.19E+06	1.42	31.93	137.1
Algorithm 11	2.32E+07	4.295	0	6.90E+03	3.13E+04	357.9	0.0001593	0.4817	0.1967
Algorithm 12	2.32E+07	4.295	0	6.90E+03	3.13E+04	2.50E+04	0.01114	0.271	1.094
Algorithm 13	2.32E+07	4.295	0	6.90E+03	3.13E+04	4.36E+03	0.001941	0.2222	0.3918
Algorithm 14	2.32E+07	4.295	498.9	6.90E+03	3.13E+04	1.64E+06	0.7288	17.76	70.59
Algorithm 15	1.59E+07	1	531.7	4.74E+03	2.13E+04	3.65E+03	0.001624	0.2961	0.2961

Fig. 8: Representation of Algorithm Summary table available on TriggerCostBrowser website containing details about the algorithm executions. They include number of events in which algorithm was activated, number of algorithm calls per event, rate of algorithm calls, rate of events in which algorithm was executed, algorithm call duration or total time of algorithm execution. Created based on reprocessing 2018 proton-proton collision data with the latest HLT software. [13]

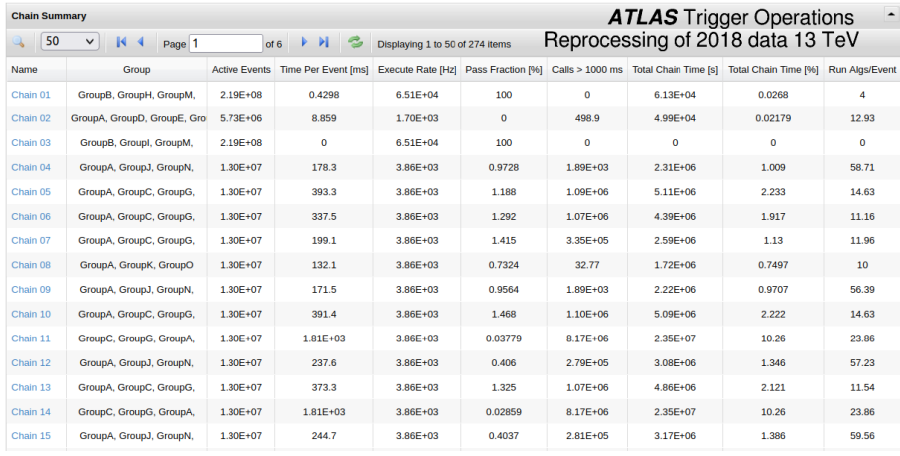
The reasons for poor algorithm performance could be multiple algorithm calls per event or the time spent waiting for data caused by a high rate of data requests from the read-out system. The number of chains that executed the algorithm should be also checked. A chain-algorithm mapping available on the website can be used to directly access associated chains' summaries. Naturally, some of the algorithms will be more expensive than the others, the computational cost of each selection must be balanced against the available resources and the ATLAS collaboration's physics goals.

Another important summary to take into the account performing cost analysis is the "Chain Summary" containing details about chains' execution, including:

- Number of events in which chain was active,
- Chain's execution rate,
- Mean time spent for this chain per event,
- Total chain execution time and what fraction of total HLT walltime it is,
- How many algorithms were called by chain on average.

The table also shows information about the chain's group, which tells the signature the chain belongs to. Using that feature signature experts can analyze the

cost of chosen groups. An example of "Chain Summary" table is presented in the Figure 9.



Chain Summary

ATLAS Trigger Operations
Reprocessing of 2018 data 13 TeV

50 | Page 1 of 6 | Displaying 1 to 50 of 274 items

Name	Group	Active Events	Time Per Event [ms]	Execute Rate [Hz]	Pass Fraction [%]	Calls > 1000 ms	Total Chain Time [s]	Total Chain Time [%]	Run Algs/Event
Chain 01	GroupB, GroupH, GroupM,	2.19E+08	0.4298	6.51E+04	100	0	6.13E+04	0.0268	4
Chain 02	GroupA, GroupD, GroupE, Gro	5.73E+06	8.859	1.70E+03	0	498.9	4.99E+04	0.02179	12.93
Chain 03	GroupB, GroupI, GroupM,	2.19E+08	0	6.51E+04	100	0	0	0	0
Chain 04	GroupA, GroupJ, GroupN,	1.30E+07	178.3	3.86E+03	0.9728	1.89E+03	2.31E+06	1.009	58.71
Chain 05	GroupA, GroupC, GroupG,	1.30E+07	393.3	3.86E+03	1.188	1.09E+06	5.11E+06	2.233	14.63
Chain 06	GroupA, GroupC, GroupG,	1.30E+07	337.5	3.86E+03	1.292	1.07E+06	4.39E+06	1.917	11.16
Chain 07	GroupA, GroupC, GroupG,	1.30E+07	199.1	3.86E+03	1.415	3.35E+05	2.59E+06	1.13	11.96
Chain 08	GroupA, GroupK, GroupO	1.30E+07	132.1	3.86E+03	0.7324	32.77	1.72E+06	0.7497	10
Chain 09	GroupA, GroupJ, GroupN,	1.30E+07	171.5	3.86E+03	0.9564	1.89E+03	2.22E+06	0.9707	56.39
Chain 10	GroupA, GroupC, GroupG,	1.30E+07	391.4	3.86E+03	1.468	1.10E+06	5.09E+06	2.222	14.63
Chain 11	GroupC, GroupG, GroupA,	1.30E+07	1.81E+03	3.86E+03	0.03779	8.17E+06	2.35E+07	10.26	23.86
Chain 12	GroupA, GroupJ, GroupN,	1.30E+07	237.6	3.86E+03	0.406	2.79E+05	3.08E+06	1.346	57.23
Chain 13	GroupA, GroupC, GroupG,	1.30E+07	373.3	3.86E+03	1.325	1.07E+06	4.86E+06	2.121	11.54
Chain 14	GroupC, GroupG, GroupA,	1.30E+07	1.81E+03	3.86E+03	0.02859	8.17E+06	2.35E+07	10.26	23.86
Chain 15	GroupA, GroupJ, GroupN,	1.30E+07	244.7	3.86E+03	0.4037	2.81E+05	3.17E+06	1.386	59.56

Fig. 9: Representation of Chain Summary table available on TriggerCostBrowser website containing details about the chain executions. They include groups the chain belongs to, number of events in which chain was activated, chain execution rate, number of algorithm calls that chain made and chain duration. Created based on reprocessing 2018 proton-proton collision data with the latest HLT software. [13]

The Chain Summary is created using collected details about the algorithm's execution. In the concurrent framework, chains executions overlap and they share the same algorithms. Using chain-algorithm mapping presented in Figure 10 related algorithms can be explored and the most expensive ones identified.

The Cost studies help to validate the framework by comparing expected and measured Trigger performance. Based on global mean time per event one can use the maximum input data rate to the Trigger to calculate the number of CPUs needed (in terms of total HEPSPEC performance), also taking into account processor efficiency when using a farm with mixed processor types.

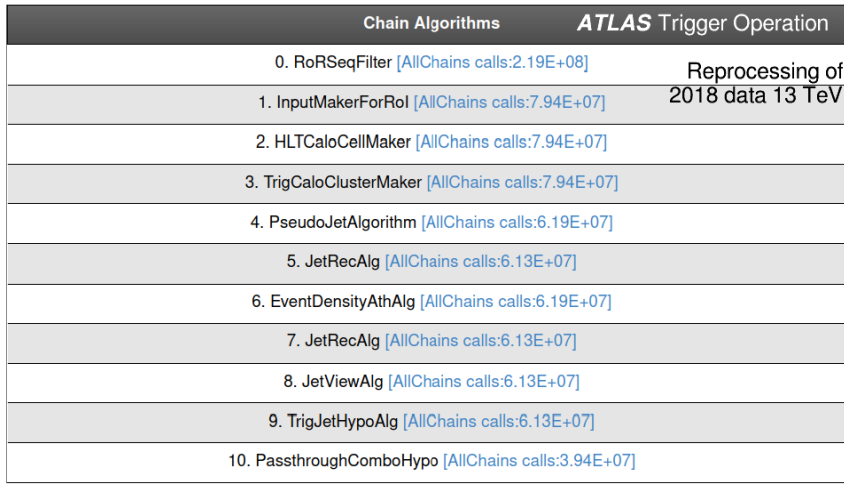


Fig. 10: Representation of Chain Item Summary on TriggerCostBrowser website that lists all algorithms related to a particular chain. In this example a jet reconstruction chain is presented. Each algorithm displays its class and the number of calls that it made ("AllChains calls"). Created based on reprocessing 2018 proton-proton collision data with the latest HLT software. [13]

REFERENCES

[1] ATLAS Collaboration: The ATLAS Experiment at the CERN Large HadronCollider, In: JINST3 (2008), S08003. doi: 10.1088/1748-0221/3/08/S08003.

[2] ATLAS Collaboration: The ATLAS Collaboration Software and Firmware, ATL-SOFT-PUB-2021-001. 2021. url: https://cds.cern.ch/record/2767187

[3] ATLAS Experiment: Approved DAQ Plots, url: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ApprovedPlotsDAQ

[4] R. Brun and F. Rademakers: ROOT: An object oriented data analysis framework, In: Nucl. Instrum. Meth. A389 (1997). Ed. by M. Werlen and D. Perret-Gallix, pp. 81–86. doi:10.1016/S0168-9002(97)00048-X.

[5] ATLAS Collaboration: Trigger monitoring and rate predictions using Enhanced Bias data from the ATLAS Detector at the LHC, ATL-DAQ-PUB-2016-002. url: https://cds.cern.ch/record/2223498.

[6] ATLAS Collaboration: Athena (2020), url: https://doi.org/10.5281/zenodo.2641996

[7] Barrand G and others: GAUDI – A software architecture and framework for building HEP data processing applications, Comput. Phys. Commun. 140 45–55. (2001)

[8] K. Rupp: Microprocessor Trend Data, url: [Online; accessed 30-08-2021] https://github.com/karlrupp/microprocessor-trend-data

[9] J. McCallum. "Memory Prices (1957-2018)". url: [Online; accessed 30-08-2021] http://jcmit.net/memoryprice.htm .

[10] ATLAS Collaboration: AthenaMT: upgrading the ATLAS software framework for the many-core world with multi-threading," J. Phys. Conf. Ser. 898, 042009 (2017)

[11] B. Hegner, P. Mato, and D. Piparo: Evolving LHC data processing frameworks for efficient exploitation of new CPU architectures, In: 2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC). 2012, pp. 2003–2007. doi: 10.1109/NSSMIC.2012.6551463

[12] A. D. Robison: Intel®Threading Building Blocks (TBB), In: Encyclopedia of Parallel Computing. Ed. by D. Padua. Boston, MA: Springer US, 2011, pp. 955–964. isbn: 978-0-387-09766-4. doi: 10.1007/978-0-387-09766-4_51. url: https://doi.org/10.1007/978-0-387-09766-4_51

[13] ATLAS Experiment: Trigger Operation Public Results, url: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/TriggerOperationPublicResults