Flexibility extension of the Event Picking Service for the ATLAS experiment

E. Alexandrov[a,b,1], *I. Alexandrov*[a,b,2], *D. Barberis*[c,3], *A. Yakovlev*[a,4]

[a] Joint Institute for Nuclear Research, Joliot-Curie 6, RU-141980 Dubna, Russia

[b] State University "Dubna", Universitetskaya 19, Dubna, Moscow region, Russia

[c] Università di Genova and INFN, Via Dodecaneso 33, I-16146 Genova, Italy

The ATLAS EventIndex is a catalog of all recorded and simulated ATLAS events, one of four main experiments at the LHC accelerator at CERN. The Event Picking Service is one of the components of the ATLAS Event Index. It is used when a user wants to collect interesting events from a huge amount of ATLAS data and reprocess them. The process of retrieving an event can be split into separate tasks. The set of tasks may differ for different event types. Some tasks use external services, which can take a long time to receive results. An error may occur as a result of completing the task. Some of these errors can be corrected automatically by the service, but some require administrator intervention. Failed tasks must be restarted from the specified step after the problem is manually corrected by the administrator. This is critical if the error occurs after a long-running task has been completed. All of the above leads to the fact that the Event Picking Service must be flexible and be able to be customized for a specific situation. This paper is dedicated to describing how elasticity is achieved in the Event Picking Service and how it improves during operation.

PACS: 07.05.Hd; 07.05.Kf

Introduction

The ATLAS experiment is one of four experiments at the LHC accelerator at CERN [1]. The ATLAS EventIndex [2], a catalog of all recorded and simulated ATLAS events, has been running since 2015 and stores data on billions of events. The main goal of the ATLAS EventIndex is to find an event by selected criteria for its further detailed analysis and/or visualization.

One of the components of the ATLAS EventIndex is the Event Picking Service. This is an automated system for extracting the requested events and delivering them to requesters. The importance of this service is especially evident when there is a need to extract many events in bulk for processing them using advanced algorithms. One example of such a need is the $\gamma\gamma \to WW$ in proton-proton interactions. The first analysis round required 50k events [3].

The main requirement for the ATLAS Event Picking, which we leave out of further descriptions, is the unconditional fulfillment of the tasks that the

[1]E-mail: aleksand@jinr.ru
[2]E-mail: alexand@jinr.ru
[3]E-mail: Dario.Barberis@cern.ch
[4]E-mail: yakovleva@jinr.ru

service faces, i.e. the implementation of search, delivery and collection of the required physical events in one file. Since this service must be able to work for many months, it must be robust. At the same time, in the event of a failure, the service must be able to restore its work. This task in one form or another is also for other high-energy physics experiments. It is necessary to take into account that even within the framework of the ATLAS experiment, the process of event extraction for different formats differs significantly. For other experiments, the implementation of the steps themselves required for the event extraction process will be completely different. Therefore, the service must be flexible, allowing its functionality to be dynamically configured depending on the required steps.

To use the service, the user will only need to provide this service with all the information necessary to find the requested events, such as a file containing the run and event numbers, the format of the requested data, and additional parameters, if any. For the ATLAS project, the additional parameters are the project name, AMI (ATLAS Metadata Interface) [4] tag, and trigger stream name.

Workflow

The Event Picking Service is not designed for one specific event extraction process, but to collect events in general. In this case, the steps required for different requests may be different. As practice shows, even during the execution of one request, there is a need to make changes to the workflow. As a result, it is important that the service be modular, and the workflow can be dynamically changed during the operation of the service.

After analyzing typical requests, two types of tasks were identified. The first type are tasks that use the full input data. Tasks of this type must be executed sequentially. The second type are those that use only part of the input. These tasks can be executed in parallel. Typically, entire chains of sequential tasks are parallel, rather than individual tasks. In this chain, a task uses all the data from the previous task, but the first task in this chain uses only part of the data from the parent task. The format of all data is JSON, which allows to dynamically change the set of input data used in the next step. In the Event Picking Service, the first type of task is called a job. Lists of sequential tasks that can be executed in parallel are called chains.

The basic idea behind each workflow is that it has an input state, an output state, and the name of the method that will perform the task. The service looks for a workflow using the input state for the selected task. Using the workflow, it dynamically runs the required code, and when completed, changes the status to the workflow's output state.

Figure 1 presents the current workflow of the Event Picking Service for the ATLAS experiment. The goal of first job is to prepare the data. In case the input data is common (a file containing the run and event numbers, the format of the requested data, and additional parameters) it only splits

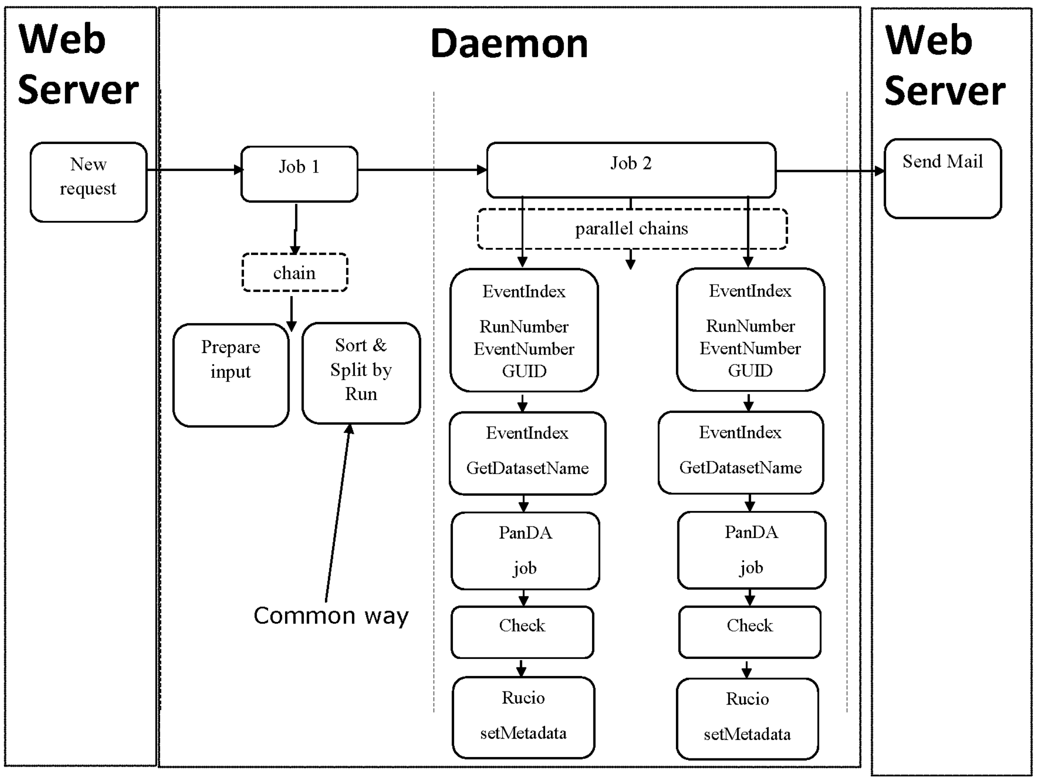

Fig. 1. ATLAS workflow of the Event Picking Service

the data into chains that will work in parallel mode. If the request uses additional workflows with alternative input data, it prepares the data using the specified method in the selected workflow. The second job involves the main work of finding and extracting events. It is usually the same for a single experiment, but for some workflows a few steps may be skipped.

In the ATLAS experiment, the following tasks are required to complete the job:

- **Get GUIDs**. The ATLAS experiment uses the Rucio [5] system to store data. The EventIndex service provides the GUID of the Rucio file for each selected run and event number.

- **Get the dataset template**. PanDA [6] uses the dataset name to start a new task, but Rucio has some rules to create new dataset names. Rucio can get the dataset name for the selected GUID. This dataset name can be used as template to start a new PanDA task.

- **Extract the events**. Extracting events from data files is a long process in common cases. The service uses the PanDA system for these long operations.

- **Check the output files**. PanDA copies the data, but the result of copying should be checked because some events may not be in the original data.

- **Push the metadata to the Rucio system**. PanDA do not set metadata of Rucio (number of events as example). This metadata should be added using the Rucio API.

During operation, some errors were detected. Some required corrections within the service itself, while others necessitated expert intervention in external systems such as EventIndex, Rucio, or PanDA. For instance, events were not indexed in the EventIndex, which meant that a reindexing process had to be initiated before tasks could be restarted within the automatic event service. Using the fact that all intermediate data is stored in the database and the ability to dynamically change data, new workflows for the ability to restart runs that ended with an error were added.

Currently the Event Picking Service has the following workflows:

- **Main**. It is main workflow to get selected events.
- **Full restart**. Restart a request using the same parameters as the previous execution.
- **Restart job on errors and warnings**. It identifies all chains that finished with an error or warning status and restarts them.
- **Restart check task**. The PanDA task requires special attention, as it can take up to two weeks to complete and usually generates a large output file. This workflow identifies all chains that concluded with an error in the check step (the step following PanDA) and restarts the job from that specific step.
- **Finalize request**. This is a specialized system workflow that is not visible to standard users. It is used in the event of a critical error in a request and sets an error status for all unfinished jobs.

Database

General architecture

As illustrated in Figure 2, the object model of the Event Picking Service contains four main elements: "request", "job", "chain" and "task". The structure can be represented as a tree, with "request" as the root and "task" as the last leaf node. Each element, except for the final one, has at least one child. The "task" object provides a description of an actual task. These elements feature three primary fields: "input", "output" and "status". The "input" and "output" fields are used to define the input and output data for each element. The real data is stored in the "storage" table, while the main elements only contain references (links) to this data. The input data for the parent and its first child is identical. Similarly, the output data for

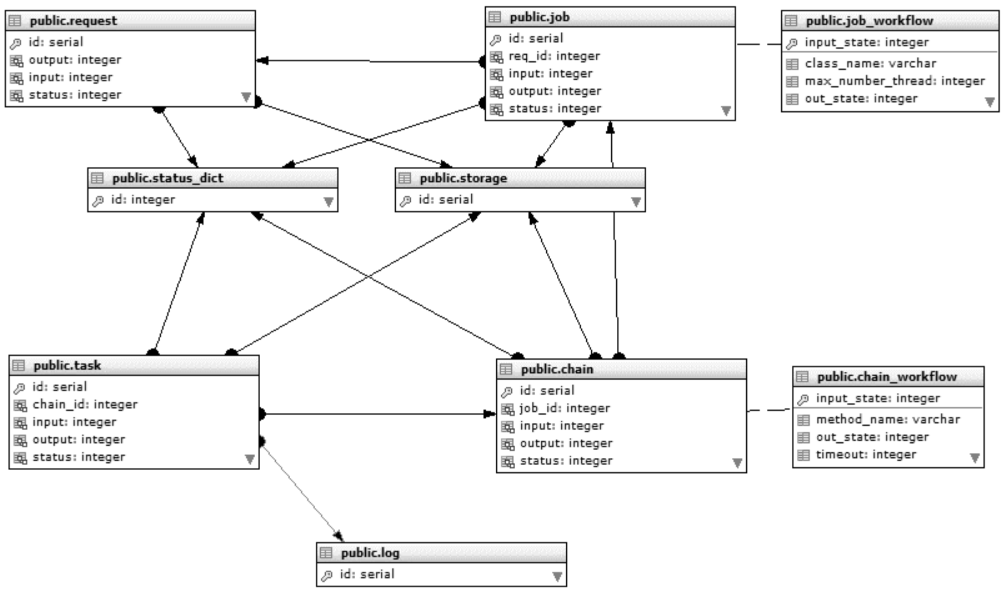

Fig. 2. The database structure of the Event Picking Service

the parent and its last child is also the same. The "status" field stores the result status and is inherited in the same manner as the output data.

The Event Picking Service comprises two types of workflows: the job workflow and the chain workflow. These workflows are stored in separate tables within the database, but the fundamental concept of constructing sequences for these workflows is consistent and is based on the input and output states. The differences that necessitate storing them in separate tables are based on additional required parameters. For example, a job creates threads and requires the specification of the maximum number of simultaneously running threads. In contrast, a chain operates within an already created thread but requires the definition of the actual executable code, such as the name of the method to be called.

A typical service architecture could include a traditional client-server model with the storage and use of necessary data in a database associated with the server. However, this system has a distinct feature. On one hand, it must be publicly accessible to users who submit their requests in the client part. On the other hand, executing these requests is associated with external systems, access to which requires a higher level of protection or may be requested by users with high privileges. Additionally, external requests often take a long time to process, sometimes reaching several days. Therefore, a logical proposal is to add another component to the architecture that handles the part of request processing associated with the execution of these requests through commands from other systems. Let's call this component a daemon, as it must operate constantly. Thus, the system should consist of three main components: a web server, a database, and a daemon.

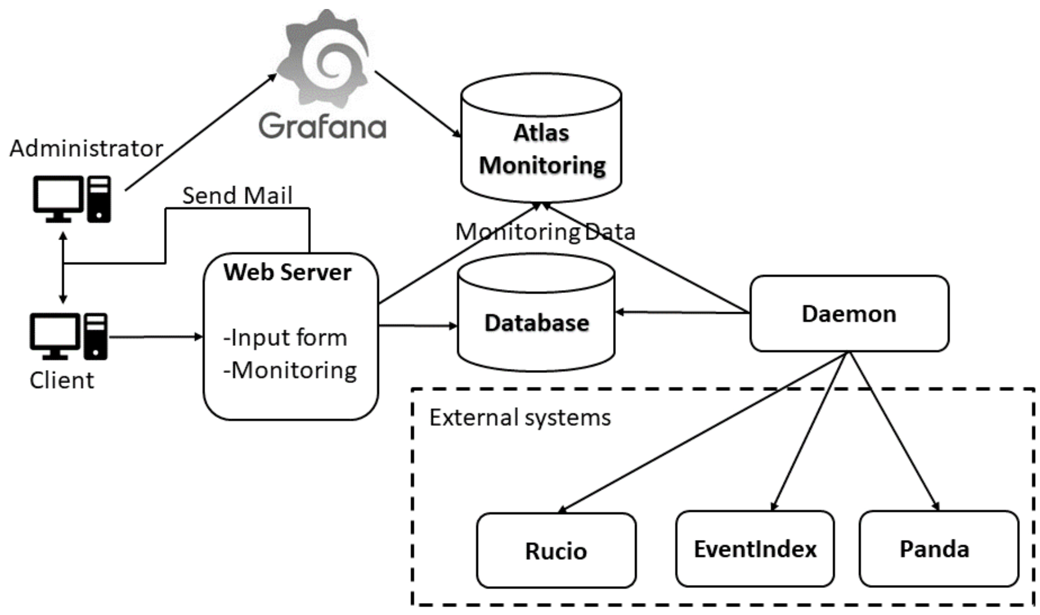

Fig. 3. The general architecture of the Event Picking Service

The general architecture of the ATLAS Event Picking Service is presented in Figure 3. The client generates a request to the server by completing a form embedded in the HTML page. Upon submission, the client receives a unique identifier for the request, which can be used to retrieve information about its status. The web server then forwards this request to the database. A daemon continuously monitors the database for new requests. When a new request is detected, the daemon initiates the execution of tasks associated with that request, following the specified workflow retrieved from the database. As these tasks are executed, input and output data, logs, and the statuses of all tasks, chains, jobs and requests are recorded in the database. This information is accessible to users and administrators through the web server. Once all tasks in the workflow are completed, or if critical errors arise during execution, the daemon updates the final status of the request. Finally, the web server generates a message relaying the results of the request to the user or administrator.

The monitoring of the Event Picking Service follows the same principles as the monitoring of other EventIndex components [7]. Both main elements of the Event Picking Service (the web server and the daemon) send monitoring data to the database every five minutes. To facilitate this, the scheduler employs a cron utility to run jobs (specifically, a Python script) at regular intervals. This Python script retrieves data from the database, verifies the timestamps of the data, and subsequently inserts it into the monitoring InfluxDB database. The viewer section utilizes Grafana's features for data visualization.

The web server is implemented on Apache Tomcat and is based on the Web Application Lego Toolkit [8]. More details can be found in [9].

Conclusions

The Event Index is one of the services of the ATLAS experiment the Large Hadron Collider (LHC). This service includes the Event Picking Service that currently operational on a production server. The exploitation of the Event Picking Service has led to enhanced error handling and the development of new workflows. Additionally, the Event Picking Service is flexible and can be adapted for use in other experiments once the necessary workflows are established.

REFERENCES

1. *Aad G. et al.* [ATLAS Collaboration] The ATLAS Experiment at the CERN Large Hadron Collider // Journal of Instrumentation. — 2008. — V. 3 S08003.

2. *Barberis D., others.* The ATLAS EventIndex: A BigData Catalogue for All ATLAS Experiment Events // Comput.Softw.Big Sci. — 2023. — V. 7.

3. *Aad G. et al.* [ATLAS Collaboration] Observation of photon-induced W^+W^- production in pp collisions at $\sqrt{s} = 13$ TeV using the ATLAS detector // Physics Letters B. — 2021. — V. 816, no. 136190.

4. *Odier J., Lambert F., Fulachier J.* The ATLAS Metadata Interface (AMI) 2.0 metadata ecosystem: new design principles and features // EPJ Web of Conferences. — 2018. — V. 214 05046.

5. *Barisits M., others.* Rucio: Scientific Data Management // Comput.Softw.Big Sci. — 2019. — V. 3.

6. *Megino F.B., others.* PanDA for ATLAS distributed computing in the next decade // J. Phys.: Conf. Ser. — 2017. — V. 898.

7. *Alexandrov E., Kazymov A., Prokoshin F.* BigData tools for the monitoring of the ATLAS EventIndex // CEUR Workshop Proceedings. — 2018. — V. 2267.

8. *Korenkov V., Kuniaev S., Semashko S., Sokolov I.* WALT PLATFORM FOR WEB APPLICATION DEVELOPMENT // CEUR Workshop Proceedings. — 2021. — V. 3041.

9. *Alexandrov E., Alexandrov I., Barberis D., Prokoshin F., Yakovlev A.* Development of the ATLAS Event Picking Server // CEUR Workshop Proceedings. — 2021. — V. 3041. — P. 91–94.