Composing Distributed Services for Selection and Retrieval of Event Data in the ATLAS Experiment

Elisabeth Vinek and Florbela Tigue Aires Viegas for the ATLAS Collaboration
CERN, CH-1211 Genève 23
E-mail: elisabeth.vinek@cern.ch

Abstract. The ATLAS experiment uses TAGs, which are basic quantities describing events recorded at the ATLAS detector, to index and optimize the access to the full event data. TAGs allow physicists to pre-select data based on queries of any of the TAG attributes. TAGs are stored in relational databases deployed at various locations, and several distributed services are in place to access and extract them. When a user starts a request, typically several of those services need to be selected and composed in order to properly respond to the request. As there can be several deployments of the same service, differing only in Quality-of-Service (QoS) attributes, there has to be a mechanism to select one deployment for each service required. In this publication, we describe the selection model and the underlying optimization considerations. The objective is not only to optimize the access for a single user, but to give to as many users as possible efficient access to the TAGs. The approach also takes into account the dynamic aspects of the distributed TAG system, where whole sites or single service deployments can be down or decommissioned. A QoS registry has been designed and implemented, to provide the data needed as input to the optimization itself.

1. Introduction
TAGs are a derived data product in the ATLAS experiment [1], produced at the end of the data reconstruction chain. They are event-level metadata containing key quantities about events and references to the events in upstream data products. TAGs are thus a means to decide which events are of interest to a specific analysis without retrieving and opening the files that contain them, and to efficiently retrieve exactly the events of interest, and no others [2]. While an event written by the Event Filter is about 1.6 MB in size, an event in a TAG file is only about 1 kB. Making a preselection based on TAGs before actually proceeding to a specific analysis can considerably reduce the amount of data queried and transferred, and thus speed up the analysis process. TAGs contain event identification, bit-encoded trigger information, physics quantities, global quantities, detector status and quality words, and reserved words for each physics and performance group. The content of TAGs has been decided by a special task force in 2006 [3] and has slightly evolved since then.

Tags are stored as POOL[4] collections, with a physical back-end either as ROOT[5] files or in a relational database. The Tier-0 Management System [6] uploads TAG data to several Oracle databases around the world, making them easily queriable through services accessing the databases. An advantage in using databases is the ability to index the data as required, in order to allow fast and efficient queries.

Published under licence by IOP Publishing Ltd
Several services exist that allow to access and query the TAG databases and display, extract or skim the results. Each of these services covers a part of the functionality, leading to a modular, service-oriented architecture. To respond to a typical request, several services have to be composed to form a workflow. Some of these services are deployed at several locations, i.e. there exists one or more deployments per service. As a consequence, for each step in the workflow, a concrete deployment has to be chosen from all possible ones. All deployments of a service are identical in terms of functionality, but they have different Quality of Service (QoS) attributes, because they are running on heterogeneous machines at various sites. Instead of randomly choosing a deployment, or hard coding deployment ranks, more sophisticated and automated strategies can be applied to optimize the deployment selection, based on user and system requirements. In order to be able to adopt an automatic deployment mechanism, several requirements have to be fulfilled:

- All system components (sites, resources, deployments etc.) have to be known and documented,
- the system has to be monitored and QoS attributes gathered, and
- the optimization objectives have to be clearly defined.

In this paper, we present how these requirements are addressed in the TAG system, and how the deployment selection challenge is approached.

The organization of this paper is as follows. In Section 2 the TAG system is introduced in more detail, focusing on deployments distribution and the objectives of an automated selection approach. Section 3 presents how the components of the distributed TAG system are modeled and documented. In Section 4, the QoS attributes gathered through system monitoring are discussed. The implementation of QoS registry and the deployment selection mechanism are presented in Section 5. Finally, Section 6 concludes the work.

2. TAGs: ATLAS Event-Level Metadata

2.1. TAG Services

In [7], the role of the TAG database (and more generally the TAG system) is described as "to support seamless discovery, identification, selection and retrieval of ATLAS event data held in the multipetabyte distributed ATLAS Event Store." TAGs allow to preselect interesting events for further analysis on the basis of the contained variables. From the selected events physicists can then navigate to the corresponding Analysis Object Data (AODs), Event Summary Data (ESDs) or RAW data, to perform their specific analysis. As this analysis can be very resource intensive, it is preferable to run it only with events that are known to satisfy basic query predicates. The main TAG use cases can be summarized as follows:

(i) Interactive queries, browsing through the TAG data using a web interface.
(ii) "Event picking": getting AOD/ESD/RAW references, with providing a list of (run number, event number) pairs as input.
(iii) Create a custom TAG ROOT file, containing only events satisfying a provided query.
(iv) Create a skimmed AOD/ESD ROOT file, also only containing events satisfying a provided query.

These use cases are addressed by the following services:

**TAG Database** is the source of relational TAG data. It is a multi-Terabyte data store [8]. The TAG database is considered as a service allowing basic access to TAG data.
iELSSI, the "TAG browser", is a web interface implemented in PHP and AJAX technologies [9], that allows to browse TAG data. It requires a connection to one or more TAG databases and allows building queries, making event counts, and displaying results and histograms.

Extract allows to produce a TAG ROOT file containing only those events that pass a given query [10]. It can be invoked from iELSSI – taking the query defined directly in iELSSI as input – or from a command-line interface.

Skim allows to produce an AOD or ESD file containing only events satisfying a given query over TAGs without having to know about the physical location of the corresponding event data. The produced skimmed file can then be used as input to run detailed analysis [11].

Event picking allows to get pointers to specific events of interest in RAW, ESD and AOD files, based on (run number, event number) pairs as input.

The above described services partly build on further, more basic building block services, such as trigger decoding, histogram and the Athenaeum service [10]. There is an ongoing effort to build a service-oriented architecture with loosely coupled services that can be distributed.

2.2. TAG Deployments Distribution and Resulting Challenges
There are currently five ATLAS sites hosting one or more TAG services, and more sites are interesting in joining. The TAG database is deployed at all sites, other services are deployed on a subset of the sites. We are thus facing a system in which a given functionality (service) exists as one or more deployments at geographically distributed locations. For the main scope of this work, the number of atomic services is not that relevant, but an automatic service selection and composition is most beneficial as the number of services and deployments increases.

| Querying and extracting TAG data |
|---|
| **User View** |
| Select set of data | Define query | Count events | Extract data | Extracted File |
| **Service View** |
| iELSSI | TriggerDecoder | TAG Database | Extract | Extracted File |
| **Deployment View** |
| Invoke Deployment d(iELSSI) | Invoke Deployment d(TriggerDecoder) | Invoke Deployment d(TAG Database) | Invoke Deployment d(Extract) | Extracted File |

**Figure 1.** Example TAG Workflow

Figure 1 shows an example TAG Workflow from three perspectives. Let’s assume that a user wants to use the TAGs to extract a ROOT file containing selected events. In this case, the user will first select a data project or period, then define a query on this data, count the resulting
events and finally extract the data in a file. Some of these steps can of course be repeated, for example refining the query, but for simplification this is not taken into account here. From the system perspective, four services are needed to complete such a request – iELSSI, Trigger decoder, TAG database and Extract. Note that there is no vertical correspondence between activities in the user view and those in the service view. Finally, to instantiate the workflow, one deployment for each service has to be selected - the \( d(\text{Service}) \) function maps exactly one deployment to the service.

All deployments of a service are identical in terms of functionality, therefore the selection is based on QoS considerations. The problem is addressed from both the user and the system provider perspective, resulting in two objectives underlying the deployment selection:

(i) On one hand, in order to provide a good quality of service, it is desirable to minimize the overall response time for each request.

(ii) On the other hand, incoming requests have to be distributed in order to allow load balancing, make an efficient use of all available resources and ensure a fair system usage for all users.

These two objectives are competing, and the resulting challenge is a multi-objective optimization problem. As TAG services are getting used more frequently as data taking at ATLAS is a continuous process and an important amount of data is available for analysis, a careful selection of services on a per-request basis is becoming crucial in order to ensure an efficient event selection system based on TAGs.

QoS-aware Service selection, and especially web service selection, is a widely studied research topic, and several approaches have been proposed [12, 13, 14]. The common first step is the design and implementation of a registry allowing to lookup for existing deployments and their characteristics. The TAG registry design is discussed in the next section.

3. Components of the Distributed TAG System

For building the model of the TAG system, the entities common to most distributed, service-oriented systems have been identified and the interactions between them have been characterized. Figure 2 shows the final model represented as a UML class diagram that has been developed and described in detail in our previous work [15].

![Figure 2. Components of the System Ontology.](image-url)
A System Ontology for the TAGs has been introduced to set a common understanding of the building blocks of the system. The distributed TAG system can be modeled as a set of physical Components and Attributes describing those components. A component can be a Provider, a Resource, a Service or a Deployment. A provider maps to an ATLAS site. A Resource is a physical or logical building block, such as a database server, a web server or a virtual machine, hosting at least one service. Each resource belongs to a provider, and a provider hosts at least one resource, otherwise it would not be part of the considered system. A Service is a generic functionality captured in a logical building block and exposed to the users. The union of all services of a system describes its functionality. Examples of concrete instances of the Service class in the TAG use case include the TAG database, iELSSI and Extract. Finally, a deployment refers to a concrete instance of a given service on a given resource. It is thus modeled as an association class between the classes Service and Resource. A Deployment is directly accessible to the user community over the Internet or a dedicated network. The Link class defines connections between components, generally referring to network links. Those links can be defined at the provider level, but also at the resource or deployment level, depending on the desired granularity. In the TAG system, we start by defining links between sites.

Component attributes can be functional or non-functional. Functional attributes describe the specific function of a component, specifying what the specific component is supposed to accomplish, as opposed to how. Examples of a functional attribute are access patterns and details of business logic. Functional attributes will typically be related to services. Non-functional or QoS attributes describe characteristics related to the operation of components, specifying how a component accomplishes a given functionality. Examples for non-functional attributes include performance metrics, availability and reliability. QoS attributes can be further classified into several categories, reflecting their different types and relations. In this work, we focus on QoS attributes.

4. QoS Attributes of TAG Components

QoS attributes are gathered through active and passive monitoring. Active monitoring means for instance that ping requests can be regularly sent to resources in order to check their availability. Passive monitoring refers to logging activity. Each deployment execution is logged in terms of data access and performance. Execution times for a similar query on several databases for instance allow to compare the query performance. This logging has to be evaluated over time. Each component identified in the previous section can be associated with specific QoS attributes.

• Attributes of deployments:
  
  **Computed Deployment Performance** is an aggregated performance measure calculated based on logging information on past deployment executions, and whose computation depends on the service. Let’s consider two examples: (1) For assessing the performance of the TAG databases deployment, the time for building local indexes on a given set of TAG data can be taken as performance benchmark measure. (2) For the Extract service, an aggregate performance metric can be derived from the logging of past executions that includes the number of events processed and the total processing time. For other services, such as iELSSI, computing an aggregate performance metric is more difficult, and basic resource attributes can be taken into account instead. In the deployment selection process, better performing deployments are preferred to worse performing ones.

  **Availability** refers to the relative up-time of the deployment.

• Resources can have static and dynamic attributes. Examples for static resource attributes include CPU count, CPU speed, maximum disk space and input/output rate. An example for a dynamic resource attribute is the current server load. Static attributes can be easily determined and entered into the QoS repository. However, at this stage of the work, a single resource metric is taken into account:
Current load refers to the measure of the amount of work that a resource is performing during the last $x$ minutes.

- **Attribute of links.** As described in Section 3, links are special system components, defining a connection between two services. When composing deployments, link metrics are very important to consider. If the service selection is optimized to choose the best-performing ones, but the network links between them are weak, the overall selection might not be optimized. Link attributes refer to network metrics.

  **Computed link performance** is an aggregated performance measure calculated based on round-trip-time measures and known network bandwidth. The round-trip-time is the network latency from source to destination and back. It can be easily determined with a ping-like command. **Network Bandwidth** is a bit rate measure of available data communication resources expressed in bits per second. In the deployment selection process, better performing links are preferred to worse performing ones.

- **Attributes of services** do not refer to measured or observed QoS values, because a service is an abstract concept. They are defined as **weights** expressing how important one QoS measure is for a given service, relative to other QoS measures.

  **Latency Sensitivity** states how latency-sensitive the relation between two services is. The latency sensitivity is usually high if there is no or only a small amount of data sent between two services, or between a service and the user. If there is no or little bandwidth required, the latency is the main factor determining the speed of a link.

  **Bandwidth Sensitivity** states how bandwidth-sensitive the relation between two services is. As opposed to the latency sensitivity, the bandwidth sensitivity is high if there is potentially an important amount of data transferred between two services. In this case, the latency is only a small percentage of the total time spent on the link and thus not as important as the bandwidth.

  **User Satisfaction** determines how important the speed between two services is, from the user perspective. For interactive services, such as querying the TAG database on the web, making rapid counts and histograms to show a few results (iELSSI browser) the perceived speed is crucial for the acceptance of the service – i.e. the user is typically not tolerant with delays and important loading times, starting in the order of seconds. On the other hand, for asynchronous jobs, such as extract or skim, a few seconds more or less do not matter that much in terms of user acceptance. For these calls, users usually get email notification upon completion and do not actively wait for a result being displayed instantly.

In order to be included in the deployment selection process, these QoS attributes and weights have to be stored in a QoS registry.

5. **Registry and Deployment Selection Implementation**

Based on the generic model presented in Section 3 and the attributes introduced in Section 4, a database schema has been developed that maps all components of the TAG system to a class of the system ontology, and allows for documentation and monitoring. This schema is referred to as the **TAG Service Catalog**. Figure 3 shows a slightly simplified design of the TAG service catalog. There are more tables than the ones represented, including tables for the links, compatibility tables, logging etc.

Each row in the F\_DEPLOYMENT table maps to an instance of the class Deployment in Figure 2. A deployment is a concrete implementation of a service (stored in D\_SERVICE) and always runs on exactly one resource from the D\_RESOURCE table. Each resource is owned by a provider or site, as in D\_SITE. All table labeled with _LOGGING store changes occurring in attributes of the
After the data catalog – that automatically keeps track of TAG data movements and thus makes data distribution details transparent to the users – the service catalog is the second step towards providing a real distributed TAG system. Each deployment has a status (all status values are stored in **D_DEPLOY_STATUS**), depending on which it is included in, or excluded from, the selection process. Deployments that are in a scheduled or unscheduled downtime at the time of an incoming request are not taken into account. This allows for an automated fail over mechanism. All status changes are logged in separate tables, in order to be able to determine the availability of a deployment. In addition to the schema shown in Figure 3, several logging tables have been put in place that are populated by the TAG deployments with information on the queries (full logical query, number of events processed etc.), users, successes/failures, execution times, links to further log files etc. This data is referred to as raw logging information. Regular database jobs run on this data. They aggregate the raw logging and compute metrics as defined in the previous section. For instance, the computed deployment performance is be approximated based on index query and execution time information. The population of the logging and service catalog tables is done via a series of HTTP interfaces to the catalog database.

Additionally, the service catalog helps to improve the software management by allowing deployment versions, version updates and compatibilities between specific deployments to be specified. It has to be investigated up to which point the population and updates of the tables can be automated, in order to assess the management overhead.

Having the service catalog in place as a component- and QoS registry, the stored information can be directly used in the deployment selection process. Referring back to the schematic

---

**Figure 3.** Overview of the TAG Service Catalog.
workflow in Figure 1, when a user is starting a request with a given set of input data, the QoS registry is queried to extract the relevant attributes. The possible combinations of deployments are evaluated in terms of their expected performance, by considering the computed performance of the deployments and links, the current load on the resource hosting the deployment, and the respective weights and user satisfaction metrics. From the user perspective, the resulting expected overall execution time has to be minimized. Because each execution is logged into the service catalog and analyzed, a resource usage profile can be established. This is the main factor for the second optimization objective, namely the distribution of requests aiming at a load-balanced system, and an even and fair resource usage.

While the work presented in this paper is focused on the deployment selection objectives and needed registries, ongoing work is concentrating on the mathematical model behind the stated optimization problem, and on algorithmic approaches [18]. In future work, we plan to include workflow patterns into the model. While in Figure 1 the represented workflow is a simple sequence, some use cases can require parallel or exclusive structures. These structures or patterns have to be taken into account when aggregating the QoS measures for the overall process.

6. Conclusion
A model for categorizing the TAG system components and assessing their QoS aspects has been developed, and a database schema has been implemented for logging the system activity and aggregating raw logging data to compute defined QoS attributes. There is currently a prototype of this schema called the Tag Service Catalog. Experience from its usage in production will be presented and discussed in future work. Two objectives for a deployment selection process in the context of the TAG system have been defined, one from the user and one from the system provider perspective. Algorithms for solving the resulting multi-objective optimization problem are under investigation and are currently prototyped and tested. An automated deployment selection process based on QoS requirements and objectives is crucial as the use cases and the number of users of the TAG system increase. It ultimately allows to integrate the various resources and deployments into a single self-optimizing distributed system, transparent to the users.

References
[1] ATLAS Collaboration, The ATLAS Experiment at the CERN Large Hadron Collider, JINST 3 S08003, 2008
[2] The ATLAS Collaboration 2005 ATLAS Computing, Technical Design Report
[3] Assamagan K E, Cranshaw C 2006 Report of the Event Tag Review and Recommendation Group http://cdsweb.cern.ch/record/942275/files/
[4] Düllmann D 2003 The LCG POOL Project, General Overview and Project Structure Proceedings of 2003 Conference for Computing in High-Energy and Nuclear Physics (CHEP 03), pp. M0K007
[5] Antcheva I et.al. 2009 ROOT – A C++ framework for petabyte data storage, statistical analysis and visualization Computer Physics Communications Volume 180, Issue 12, pp. 2499-2512
[6] Elsing M, Goossens L, Nairz A, Negri G 2010 J. Phys.: Conf. Ser. 219 072011
[7] Cranshaw J, Goossens L, Malon D, McGlone H and Tique Aires Viegas F 2008 J. Phys.: Conf. Series 119 072012
[8] Viegas F et al 2010 J. Phys.: Conf. Series 219 072058
[9] Zhang Q 2010 Engineering the ATLAS TAG Browser J. Phys.: Conf. Ser. accepted for publication.
[10] Hrvnac J 2010 ATLAS Tags Web Service calls Athena via Athenaeum Framework J. Phys.: Conf. Ser. accepted for publication.
[11] The Skimming Service - https://twiki.cern.ch/twiki/bin/viewauth/Atlas/SkimmingService
[12] Alrifai M, Risse T 2009 Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition Proceedings of the 18th International Conference on World Wide Web ACM, pp. 881–890
[13] Liu Y, Ng A, Zeng L 2004 QoS Computation and Policing in Dynamic Web Service Selection Proceedings of the 13th International World Wide Web conference ACM, pp. 66–73
[14] Jaeger M, Mühl G 2007 QoS-based Selection of Services: The Implementation of a Genetic Algorithm *KiVS 2007 Workshop: Service-Oriented Architectures and Service Oriented Computing (SOA/SOC)* pp. 359–370

[15] Vinek E, Beran P, Schikuta E 2011 Mapping Distributed Heterogeneous Systems to a Common Language by Applying Ontologies *PDCN 2011: Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Networks* accepted for publication

[16] Zeng L, Benatallah B, Dumas M, Kalagnanam J and Sheng Q 2003 Quality driven web services composition Proceedings of the 12th international conference on World Wide Web (NY, USA: ACM) pp 411–421

[17] Jaeger M C, Rojec-Goldmann G, Muhl G 2004 QoS Aggregation for Web Service Composition using Workflow Patterns Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference Washington, DC, USA, September 2004, pp. 149159

[18] Vinek E, Beran P, Schikuta E 2011 A Dynamic Multi-Objective Optimization Framework for Selecting Distributed Deployments in a Heterogeneous Environment *The International Conference on Computational Science*, accepted for publication