The challenges of mining logging data in ATLAS

Elizabeth J Gallas[1,*] and *Nurcan* Ozturk[2], on behalf of the ATLAS Collaboration

[1]Department of Physics, University of Oxford,
Denys Wilkinson Bldg, Keble Rd, Oxford OX1 3RH, United Kingdom
[2]Department of Physics, University of Texas at Arlington, Arlington, Texas 76019, USA

Abstract. Processing ATLAS event data requires a wide variety of auxiliary information from geometry, trigger, and conditions database systems. This information is used to dictate the course of processing and refine the measurement of particle trajectories and energies to construct a complete and accurate picture of the remnants of particle collisions. Such processing occurs on a worldwide computing grid, necessitating wide-area access to this information.
Event processing tasks may deploy thousands of jobs. Each job calls for a unique set of information from the databases via SQL queries to dedicated squids in the ATLAS Frontier system, a system designed to pass queries to the database only if that result has not already been cached from another request. Many queries passing through Frontier are logged in an Elastic Search cluster along with pointers to the associated tasks and jobs, various metrics, and states at the time of execution. PanDA, which deploys the jobs, stores various configuration files as well as many log files after each job completes. Information is stored at each stage, but no system contains all information needed to draw a complete picture. This presentation describes the challenges of mining information from these sources to compile a view of database usage by jobs and tasks as well as assemble a global picture of the coherence and competition of tasks in resource usage to identify inefficiencies and bottlenecks within the overall system.

1 Introduction

A key component for grid-wide processing of ATLAS[1] event data is the global distribution of information from ATLAS databases which is necessary for that processing. Since the deployment of the Frontier technology[2] at grid sites early in Run 1, clients anywhere on the WLCG have benefited from efficient access to database-resident data, enabling a wide variety of event processing and analysis.

The load on the system has grown steadily over the years with the increasing rate of job submission and widening diversity of jobs. Despite continuous improvements to the system to provide greater throughput, in the last year or so, clients every so often observe increased latency in retrieving the data. In addition, there have been occasions where components of the system were overloaded to the point of failure when an appreciable number of particular types of jobs are deployed. Throttling the submission of those job types effectively alleviates

*e-mail: gallas@cern.ch

the menace of meltdown in the short term, but we need to find a way to robustly provide all task workflows with the data they need without disruption or restraint.

The overall system is measurably proficient: the Frontier launchpads adeptly handle well over 100 million queries per day and many more times that number of queries are satisfied by the caches of the Frontier squids deployed at hundreds of grid sites across the globe. We know of no single point of failure or bottleneck in the system. This presentation gives a summary of the methods and findings from selective probes of the available data logged by components of the infrastructure. We aim to better understand the SQL queries themselves and the jobs which submit them to identify inefficiencies and suggest how we can improve data storage, job configuration, and ATLAS event processing software in the future.

2 ATLAS Database Distribution Strategy

ATLAS data processing by jobs needs geometry, trigger, and conditions information which is stored in Oracle[3] Database servers at CERN. This information is made available grid-

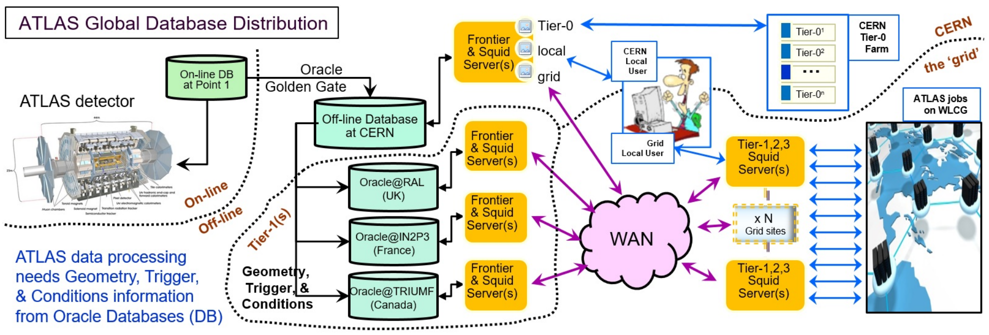

Figure 1. Schematic of ATLAS global database distribution which makes geometry, trigger, and conditions information from central ATLAS databases at CERN available to clients grid-wide.

wide as shown in Figure 1. The dotted lines of the figure delineate between the 3 distinct processing environments:

1. **Online @ CERN**:
   The online database at LHC Point 1 which is populated with configuration and conditions data needed for online data taking by the ATLAS detector.

2. **Offline @ CERN Tier 0**:
   The CERN offline database contains data needed for offline data processing and analysis. This includes a replica of some data from the online database (populated via Oracle Streams technology) and data entered by various system experts (such as offline tuned calibrations). This database is coupled with a distinct set of Frontier launchpad and squid servers (squids) dedicated to serve three specific client communities:

   - the CERN Tier 0 computing farm, whose primary role is to perform initial processing of event data from the online system
   - local users at CERN, which include a wide variety of individual system developers and analysts, automated nightly software release testing, and reprocessing of data for specialized trigger studies
   - ATLAS users on the WLCG who are closer to CERN on the WAN than to ATLAS Tier 1 sites housing the database replicas (below)

3. **Three WLCG Tier 1 centers**:
   Oracle servers at three Tier 1 sites (RAL in the UK, IN2P3 in France, and TRIUMF in Canada) each contain a replica of data needed by grid-wide processing. These databases are populated via Oracle Streams from the CERN Offline database.
   Like the CERN offline database, these databases are each coupled with one or more Frontier launchpads and squids to serve all requests coming from clients on the WLCG: ATLAS jobs deployed by Panda[4] (the ATLAS Production and Distributed Analysis system) and individual developers and analysts at grid sites.

4. **WLCG grid sites**:
   Frontier squids are installed at hundreds of grid sites. They accept queries from Panda jobs or users at grid sites: They return query results either from their cache or from the closest Frontier launchpad (noted above) to get the result to pass back to the client.

This database distribution strategy has worked well since the deployment of Frontier in Run 1. The advantages are summarized here since they factor into collective efficiency of the system:

- **Load leveling of requests:** Each launchpad and squid individually queues incoming requests (which can spike at times, such as when many jobs start around the same time). At the launchpad level, this effectively levels the rate of queries to the database. At the squid level, this quells surges in network traffic in the delivery of the response to the client.
- **Load balancing of requests at multiple levels:** Squids at many sites are groups of servers which accept requests in turn from a round-robin load balancer. Heavily loaded launchpad sites pass excess load to other sites via fail-over mechanisms.
- **Caching of results of identical queries at multiple levels:** Both launchpads and squids cache results, enabling them to return results from the cache when incoming queries are identical to those already processed.

The combined effect of the above factors is a dramatic reduction in the query load on the databases and a much lower latency in delivering results to clients especially when the cache efficiency is high. Cache efficiency in excess of 90% has been observed, but even when that is degraded, the system, as deployed, returns results to clients on average faster than a direct database query since some level of caching reduces proportionately that network traffic as well as the load on the database.

So what is the problem? The ideal levels of cache efficiency (>90% mentioned above), is observed when jobs are dedicated to tasks such as data reprocessing. In these tasks, jobs are deployed run-by-run, so concurrent jobs are processing events in the same range of runs so their queries tend to be well aligned in terms of IOV (Interval of Validity). But some types of tasks are not so regular: they require supporting data in completely different IOV ranges, which defies Frontier's cache efficiency. So the problem is that with an increasing number and diversity of clients, components of the system are strained, occasionally to the breaking point.

Clients generally requires information from the geometry, trigger, and conditions databases. The number of queries for geometry and trigger information is relatively small and the data retrieved is primarily configuration related (lower volume and more likely to be efficiently cached). The conditions database (CDB) contains a much broader range of information, for example, detector controls, alignment, LHC beam information, and data from many detector subsystems. Queries to the CDB predominate overall DB usage in intensity, diversity, and volume and has been found to be associated with degradation and failures of the distribution infrastructure. Therefore, our analysis described here focuses specifically on the CDB access.

We need to better understand the database usage patterns of clients, which will help us globally tune the distribution infrastructure and help us work with clients to refine their requests. In the next sections, we describe our investigations into the "client side" (the database access and usage of the jobs) and the "server side" (the queries observed by the database distribution infrastructure) which each hold key information from which we can suggest how components of the system can be improved.

3 The client side

We focus on offline clients shown schematically in blue in Figure 2. Worldwide distribution

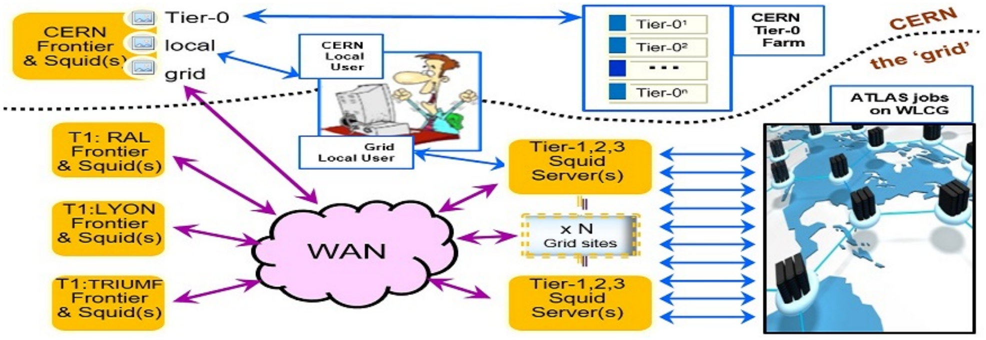

Figure 2. Schematic overview of grid-wide client (blue squares) access to the distributed network of Frontier squid servers and launchpads (orange stadiums) over the local (blue arrows) or wide (purple arrows) area networks. This enables any network-enabled client to get the DB information they need.

of Frontier squids ensure that most clients (at CERN or on the WLCG) have access to a squid via a LAN connection which may contain the data they need (blue double arrows).

While some clients are running non-Athena based processes, the vast majority of jobs are based on Athena [5] (the ATLAS software framework used for event processing), in which case a summary of the CDB data retrieved over the course of job execution is written into Athena log files. The summary includes an accounting of the data volume, the number rows and channels accessed, and the latency (time required) to access the data per schema (detector subsystem) and per folder (tables of conditions data within that schema). The log also contains warnings about which data were retrieved but never used by the job (i.e. wasted queries). In addition, it reports if multiple queries were required over the course of the job, which occurs when the IOV of the input events of the job surpasses the interval of the CDB data retrieved in previous queries. Any or all of these factors can be signs of inefficiency in the way the job is requesting data or in the structure of the storage of the data itself in the CDB system.

The location of the logs depends on how and where the job was submitted, so we consider clients by their location below because particular tasks are better suited to some locations than others and in each case, the log files, when they exist, end up in different logging architectures as noted below:

- **CERN-based Tier-0 farm**:
  The Tier 0 farm primarily performs Athena-based first processing of newly recorded data. Cache efficiency is high as expected due the nature of the processing: a steady progression of tasks run by run in sequence (naturally suited to the caching mechanisms of Frontier). Athena logs are initially kept on EOS, then eventually archived on CASTOR if needed.

- **Users running jobs on local machines**:
  These processes are very diverse and may or may not be Athena-based. Problematic CDB access from jobs in this category, can go noticed until the scale of the process grows to a point where it triggers server side alerts or anomalies (next section). When we become aware of a problem, we use available means to try to find the user to understand the use case, analyze their logs, and create more efficient methods. As examples:
  - **Experts developing systems:**
    Non-Athena-based development requiring DB information has greater potential for anomalous data requests since they are not regulated by Athena-based methods which by construction restrict the range of the IOV of queries to a rational size. Long running queries are glaringly evident from server side logging. Many times experts fix the problem before we manage to contact them because their program crashes or runs very slowly because retrieved data volume is much larger than they expect.
  - **Analyzers computing final luminosity:**
    LumiCalc, the ATLAS utility for computing publication-suitable recorded luminosity for specific datasets, is a CDB data intensive task requiring hours of execution time when a dataset spans a long period of time. Long-running and disconnected queries from machines dedicated to running LumiCalc have been observed in server side monitoring. Since users are running the process individually (and ad hoc), the heavy queries create an intermittent but relatively short term load on the infrastructure. Revision of the folders utilized is required to streamline this process.
  - **Users executing Athena based test jobs:**
    Generally users configure test jobs based on other processes which have been successful in the past, not noticing it is retrieving more data than it needs. These test jobs themselves generally do not cause excessive load on the system, but eventually, when tests are complete, the user will then proceed to deploy tasks at full scale on the grid using Panda (the next item). It is at this point that unnoticed inefficiencies become magnified.

- **Event processing on the grid**:
  The Panda system was developed by ATLAS to deploy data production requests on available world-wide computing resources. During CHEP 2016[4], Panda reported to deploy "typically 250,000 jobs concurrently running in the system and more than 5 million jobs are processed in total per week".

  Panda tasks are generally composed of many (hundreds) Athena-based jobs running over the event files of the input dataset being processed. Each job typically executes a few sequential stages of processing, each requiring specific CDB information to proceed. Each stage will write a separate Athena log file and the Panda system nicely makes these log files available for debugging and analysis.

Given the scale of jobs deployed by Panda (relative to other processes), we know the majority of the load on Frontier is collectively from jobs submitted by Panda. But there hundreds of thousands of jobs running per day; each job contributes to the load on Frontier but the Athena logs do not have sufficient information to determine which cause a disproportionate load. For this, we look to the "server side" described in the next section.

4 The server side

As shown in Figure 2, any network-enabled job can get the DB information they need via the distributed Frontier architecture. Clients issue queries to their local squids; if results are not found in the squid cache, the request is passed along a chain of squids across the WAN toward a Frontier launchpad which queries its local database if needed. The launchpad returns the result to clients via squids, each of which caches the results to satisfy future identical requests.

All launchpads and squids log every transaction into local log files. These files collectively encapsulate not only all client requests but also all interactions with the database. The logs are simple ASCII files recording every query with a timestamp with additional information depending on a variety of circumstances and outcomes: associated metrics related to the retrieved data (e.g. rows, volume, etc.) and to infrastructure performance (e.g. whether there was a cache hit or not, transaction and query time, various flags reflecting modes of success or failure). The information logged may also include identifying information about the client submitting the request. Globally, launchpads handle over 100 million requests per day and many more times that number of queries are logged by the squids deployed at hundreds of grid sites across the globe. These log files grow very quickly so sites must delete the logs regularly (generally keeping only the previous several days of information).

In years past, the logs were accessible only at the site and due to their verbose nature, size, and format, they were difficult to mine effectively even during specific time intervals. To facilitate the monitoring of the overall infrastructure, information from Frontier logs at selected sites have been extracted into ElasticSearch[6] (ES) repositories as described elsewhere in this conference[7]. Figure 3 shows schematically the processes by which the log data is collected, filtered, and stored in ES. In this system, Filebeat[8] collects log lines locally, sending

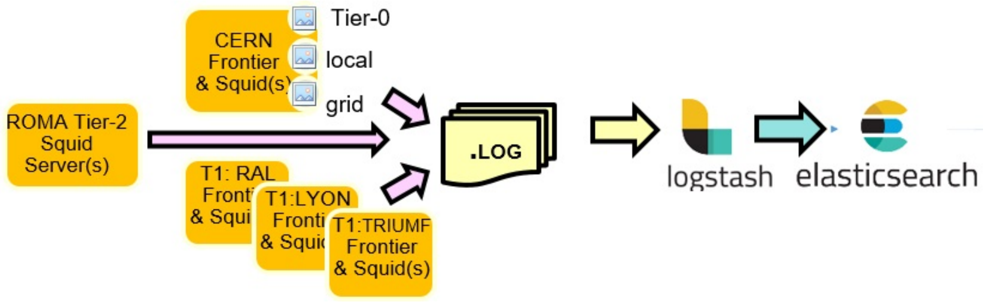

Figure 3. Schematic overview of the collection of information from Frontier logs into a centralized ElasticSearch repository. The logs of CERN and Tier 1 launchpads as well as logs from one Tier 2 squid undergo this collection process which facilitates monitoring of the overall infrastructure as well as provides critical information for our dedicated studies of particular workflows.

them to a dedicated system where Logstash[9] filters them to extract the key information to be registered into the ElasticSearch repository. As described in Reference [6], the ES repository is used to facilitate monitoring of the overall infrastructure and, for example, to identify problematic tasks which need to be throttled by Panda.

In this presentation, we use the information in the same ES repository to study in detail the requests of particular tasks and use client identifying fields in ES to cross-reference with the corresponding job log files in Panda or from particular users described in Section 3 (the "Client" side).

5 Summary

We have described two large repositories from which we use information to study client database usage across the grid.

- **Athena log files: represent the "client-side"**:
  PRO: Each Athena job log provides a tabulation of database row and channel counts accessed, associated Athena data volume, and latency in data retrieval per database schema

and folder accessed per stage in the course of the execution of each job. It also contains warnings about data retrieved but not used by the job and how many times a folder was re-queried to retrieve a new IOV.
CON: It does not contain individual SQL queries or about the uniqueness of its requests in comparison to other jobs in the same task or in other concurrent tasks. Its measured database access latency tends to reflect the load on the distribution infrastructure or cache efficiency of the given query at the time rather than the efficiency of the query itself.

- **ElasticSearch repository: represents the "server-side"**:
  PRO: Information in ES from Frontier launchpads include all interactions of those launchpads with the database at those sites (at CERN and the database replicas at 3 Tier 1 sites) and include requests passed from squid servers for which the result was not already cached.
  CON: While it may know the client task and jobs identifiers, it has no knowledge of the type of job, which stage of the job made the request, or whether the data was used by the job (or not). Moreover, it generally will not contain all requests of any particular job since the request may have been satisfied by a squid whose logs were not collected in ES (except for two site squids where the dedicated stress tests run).

These “Big Data” sources are complementary, with considerable unique and essential information to form micro- and macroscopic views of usage patterns and causes of degradation in the infrastructure. Because neither contains all the information needed we have begun to consolidate information from these sources along with auxiliary information about folders and schemas from collected metadata in COMA[10] (collected metadata about the Conditions DB) as described separately in this conference[11].

Studies thus far have concentrated on tasks deployed in dedicated stress tests with problematic workflows known to cause excess load on the distribution infrastructure, tasks identified in ES alerts, and tracking down clients which submit anomalous SQL requests. The results so far have enabled us to bring to light many underlying issues which need to be addressed:

- Generally all tasks request some data which is never used by the job. This is generally a job configuration issue, but sometimes is a deeper issue within the software.
- The same job deployed in different ways where DB access should be identical are not, for example in running Athena in single or multi-core processor modes.
- The storage model used by some folders requires revision: The original design was optimized for data insertion rather than for offline read-access.
- Some folders contain an inordinate number of IOVs. A smoothing of the data in wider time intervals would improve both query efficiency and reduce job memory requirements.
- Ill-formed queries or requests for large data volumes have also been identified, generally from developers using non-standardized methods or are simply unaware of folder content.

The issues noted above came either as a surprise that they would exist or the extent to which they exist. Each adds an unnecessary load to the infrastructure especially when magnified at the task level (since tasks deploy potentially hundreds of jobs). Eliminating or moderating these effects will require effort from many communities: Core software, job configuration, task deployment, and conditions database experts as well as work from subsystem experts who are knowledgeable about how their conditions are inserted in the database and how that data is used by offline processing. A number of initiatives are already underway which are expected to improve the efficiency of DB access: A complete overhaul of ATLAS job configuration is underway[12] and planned evolution of ATLAS conditions storage and access[13]. We aim to eventually provide more automated tools for evaluating new software[14] and workflows to catch sources of inefficiency before they are deployed full-scale.

6 Conclusions

In the past, the ATLAS database distribution infrastructure was operating well below capacity so database queries from clients in excess of requirements were not noticed. But as the number and diversity of jobs requiring database access grew over the course of LHC Run 2, the infrastructure showed signs of strain under the load and some specific workflows needed to be throttled to maintain global operations.

Two 'big data' resources have been key in the investigations to gain an understanding of grid-wide client usage of database information: Frontier log information collected into ElasticSearch repositories and Athena job logs made available via Panda. Information from these large repositories, in combination with an understanding of the conditions database structure and content have been used to form micro- and macroscopic views of usage patterns and sources of abnormal or excess load on the global database infrastructure.

Studies using this data so far have shown we must log, study, and report our findings in coherent ways to form the motivation to improve current and influence future conditions storage and access patterns. While we work with experts in software, job configuration, database, and subsystems to address immediate issues for the short term, we are aiming to eventually provide tools to regularly measure the database-wise efficiency and load grade of any task.

References

[1] The ATLAS Collaboration (2008) JINST **3** S08003.

[2] Barberis D et.al. (2012) *"Evolution of grid-wide access to database resident information in ATLAS using Frontier"*, J. Phys.: Conf. Ser. **3**96 052025.

[3] Oracle Database, http://www.oracle.com

[4] Barreiro Megino F H et.al. (2017) *"PanDA for ATLAS distributed computing in the next decade"*, J. Phys.: Conf. Ser. **8**98 052002.

[5] Calafiura P et.al. (2015) *"Running ATLAS workloads within massively parallel distributed applications using Athena multi-process framework (AthenaMP)"*, J. Phys.: Conf. Ser. **664** 072050.

[6] Elasticsearch: https://www.elastic.co/products/elasticsearch

[7] Svatos, M et.al. *"Understanding the evolution of conditions data access through Frontier for the ATLAS Experiment", EPJ Conf. for CHEP'18 (to be published).*

[8] Filebeat: https://www.elastic.co/products/beats/filebeat

[9] Logstash: https://www.elastic.co/products/logstash

[10] Gallas E J, Albrand S, Borodin M, Formica A (2014) *"Utility of collecting metadata to manage a large scale conditions database in ATLAS"*, J. Phys.: Conf. Ser. **513** 042020.

[11] Rinaldi L, Gallas E J, Formica A (2018) *"Optimizing access to conditions data in ATLAS event data processing", EPJ Conf. for CHEP'18 (to be published).*

[12] Lampl W (2018) *"A new approach for ATLAS Athena job configuration", EPJ Conf. for CHEP'18 (to be published).*

[13] Rinaldi L, Formica A, Gallas E J, Ozturk N, Roe S (2018) *"Conditions evolution of an experiment in mid-life, without the crisis (in ATLAS)", EPJ Conf. for CHEP'18 (to be published).*

[14] Leggett C, Shapoval I, Snyder S, Tsulaia V (2018) *"Conditions data handling in the multithreaded ATLAS framework", EPJ Conf. for CHEP'18 (to be published).*