The ATLAS Wide-Range Database and Application Monitoring

Petya Vasileva[1,*], *Andrea* Formica[2,**], and *Gancho* Dimitrov[1,***] on behalf of the ATLAS Collaboration [****]

[1]European Organization for Nuclear Research (CERN)
[2]Université Paris-Saclay, IRFU/CEA (FR)

Abstract. In HEP experiments at LHC the database applications often become complex, reflecting the increasingly demanding requirements of the researchers. The ATLAS experiment has several Oracle DB clusters with over 216 database schemes each with its own set of database objects. To effectively monitor them, we designed a modern and portable application with exceptionally good characteristics. Some of them include: A concise view of the most important DB metrics; a list of top SQL statements based on CPU, executions, block reads, etc.; volume growth plots per schema and DB object type; a database jobs section with signalization for failures; and in-depth analysis in case of row-lock contention or DB sessions.
This contribution also describes the technical aspects of the implementation. The project can be separated into three independent layers. The first layer consists in highly-optimized database objects hiding all complicated calculations. The second layer represents a server providing REST access to the underlying database backend. The third layer is a JavaScript/AngularJS web interface. In addition, we will summarize the continuous integration cycle of the application, which uses GitLab-ci pipelines for basic testing, containerization and deployment on the CERN Openshift infrastructure.

1 Introduction

In the ATLAS experiment[1] we have a wide spectrum of database applications dealing with data processing and bookkeeping as well as data, such as event meta-data, detector calibration constants, and detector geometry description. The data are hosted in several Oracle databases[2]. We can identify three production databases with different purposes: one related to the detector data (accessible from the network in the ATLAS experimental area), one for the offline related data and one for distributed computing data. In addition, there are two full copies of the online and distributed databases and two test databases dedicated to applications developments.

*e-mail: petya.vasileva@cern.ch
**e-mail: andrea.formica@cern.ch
***e-mail: gancho.dimitrov@cern.ch
****Copyright 2018 CERN for the benefit of the ATLAS Collaboration. Reproduction of this article or parts of it is allowed as specified in the CC-BY-4.0 license

Over 100 unique applications exist in the ATLAS databases. Some of those are spread among several database schemas, others own thousands of database objects and some execute hundreds of parallel processes per second. Specific applications such as PanDA[4], Rucio[3], WinCC[5], are of significant importance for the scientific community and any disturbance in serving their DB workload has to be avoided. Moreover, early detection of potential database issues might be important for smooth operation of the databases.

In order to perform monitoring and provide efficient support for the database schemas, a dedicated Web application was developed. The ATLAS Database & Application Monitoring (DBMon)[6] is a custom Oracle DB monitoring tool created for the needs of the ATLAS application developers and the Oracle database administrators. Its goal are to:

- present an overview of the current state of the database clusters
- provide detailed information about a selected database cluster
- produce reports based on specific database schema name

2 Architecture

The project has three independent layers, shown in Figure 1 as a Database component on the left; a Java Application component in the centre; and a web application component shown on the right. The first layer consists of highly-optimized database objects fed by sophisticated calculations in PL/SQL functions and procedures. The second layer represents a Java application providing REST access to the underlying database back-end. The third layer is a JavaScript/AngularJS web interface visualizing the data.

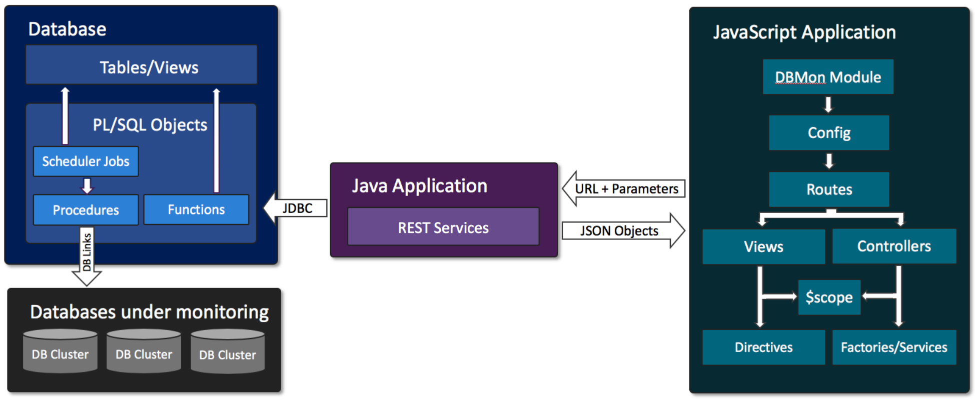


Figure 1: Database & Application Monitoring Architecture.

2.1 Database setup

The DBMon application resides at one of the ATLAS databases (the offline cluster), where a couple of scheduler jobs run regularly to collect information for the current database as well as the additional clusters. The jobs call various PL/SQL[7] procedures, which read data from multiple DB view objects (such as vsql, vsessions, v$dba_objects, etc.). Then the data are inserted into a dedicated set of tables.

The information is primarily statistical, such as the number of sessions and their distribution across the database instances; details about running/failed/succeeded jobs; a full log of currently running queries and number of executions; a list of used CPU resources; etc. The automation of the process boosts the administrators productivity and improves their efficiency. In addition, it provides valuable information to developers who work on their application database layer.

2.2 Java/REST application

The REST application is based on Spring[8] and Jersey[9], providing a number of HTTP endpoints for feeding the Web application with data. The code is minimalistic. All query calls are wrapped into PL/SQL functions, which allows easy changes when necessary. For example:

```
sql = "SELECT * FROM TABLE(atlas_dbmon.get_blocking_sessions(:db, :fromDate, :toDate))";
```

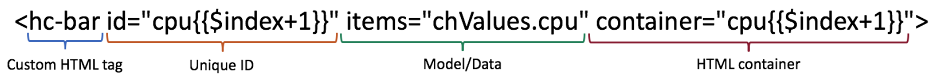
Behind *get_blocking_sessions* function there is a complex hierarchical query which uses dynamic parameters. The result is sent in the form of a simple table, whereas the logic remains hidden. Moreover, refining the query does not require a person to become familiar with the Java[10] project.

2.3 Web application

The front-end of the application is built with the AngularJS[11] framework. The technology is JavaScript-based and provides Model-View-Controller[12] design, which makes it suitable for implementing the database monitoring.

The concept behind AngularJS gives us tools for creating a well structured application, works well with the popular JavasScript[13] libraries and allows two-way data binding between the Model and the View. The "scope" is one of the building blocks of AngularJS. It forms the Model and allows the View and the Controller to access all variables in their specific scope.

Since the monitoring application is developed with the idea of being constantly extendable, we took advantage of another AngularJS feature - the directives. They let us create reusable components that could be called with a custom HTML tag and different parameters. For example, a bar-chart can be rendered by adding the following line:



```
<hc-bar id="cpu{{$index+1}}" items="chValues.cpu" container="cpu{{$index+1}}">
```

For the creation of interactive charts we use Highcharts[14]. The monitoring is dependant also on Bootstrap[15], Angular Material[16] and TreeGrid[17]. All of the libraries are open source and actively supported.

3 Features

The presented application is a tool that allows developers to explore and tune the application's database performance. As such, it provides various data representations through graphical components, which also help in detecting database issues and abnormal application behaviour. The key features are listed below:

- ***Historical plots of database activity*** - DBMon provides a set of 10 plots describing the state of the different database instances over specified period. This helps in identification of problematic database or application behaviour by narrowing down to start, end and DB instance.
- ***Top resource consumers (Fig. 2)*** - When a database experiences high load, it is crucial to efficiently understand the root cause for that. DBMon machinery captures all DB requests (among thousands per second), having longer than 3 sec execution time and stores their most important metrics. Then the SQL statements are ranked and the top 10 of them are displayed in charts as seen bellow.

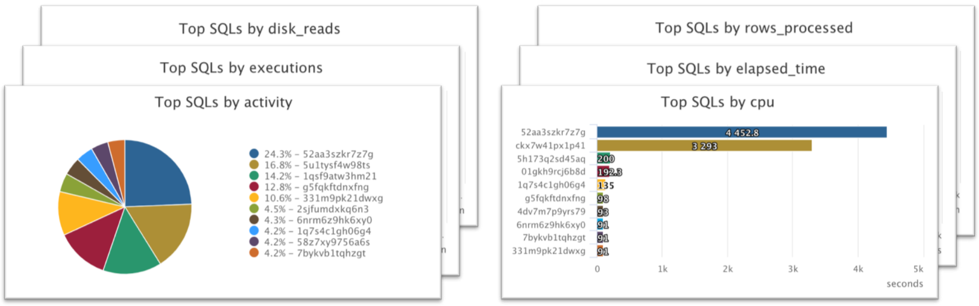


Figure 2: Top SQL queries by different criteria.

- ***Monitoring of increased values for basic DB metrics*** - It is important to keep track of a number of basic DB metrics such as "CPU Utilization", "Current OS Load", "Logons Per Sec", etc. in real time. Thus, the DBMon application provides a view of those metrics and refreshes the data periodically.
- ***DB volume growth per schema (Fig. 3)*** - ATLAS databases host several applications which data volume is growing fast and needs to be closely monitored. Furthermore, developers benefit from the fact that DBMon provides a way of plotting the size of their DB schemes.
- ***Tracking the number of active/inactive sessions per DB user (Fig. 4)*** - A common reason for increased database activity is the high number of active sessions for a specific application. Therefore, DBMon gives the number of active/inactive sessions spread among the DB cluster machines. Moreover, the monitoring provides additional in-depth details per session (e.g. SQL ID, service name, reason for waiting, etc.).

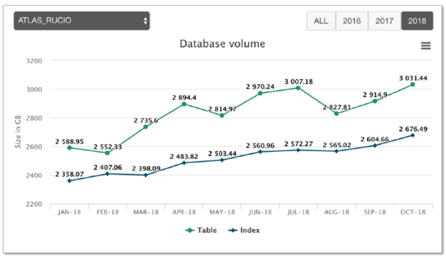


Figure 3: Rucio data volume growth for the current year.

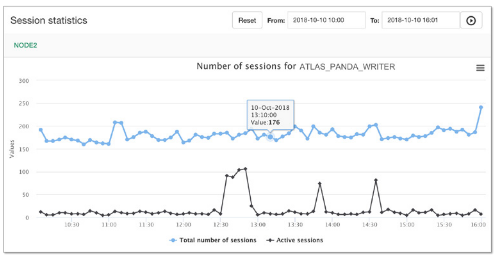
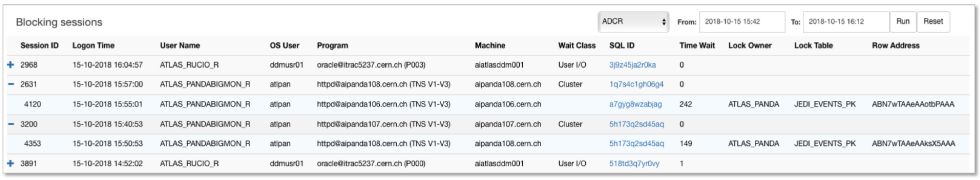


Figure 4: Number of sessions for PanDA writer account.

- ***Database jobs' state*** - Although, database jobs are not widely used by the ATLAS applications, there is number of those that run regularly on the ATLAS databases. Some of them gather statistics, others make calculations or feed other tables. It is important to have a tool that observes the state of the Oracle jobs and notify administrators when there are failures.
- ***Blocking/blocked sessions and potential reasons for the blocking (Fig. 5)*** - The ATLAS production databases serve transactional workload by hundreds of concurrent processes. In certain circumstances that might lead to row(s) lock contention or high concurrency on table or index blocks ("hot blocks"). In those cases, DBMon provides valuable information by displaying an hierarchical tree which shows unambiguously the blocker session as well as details about it. This is one of the ATLAS DBMon features that excels compared to the other monitoring tools. Moreover, blocking/blocked sessions information is stored historically for subsequent analysis.



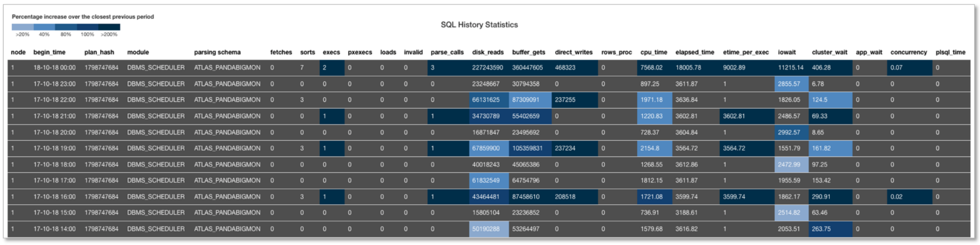
Blocking sessions

ADCR From: 2018-10-15 15:42 To: 2018-10-15 16:12 Run Reset

Session ID	Logon Time	User Name	OS User	Program	Machine	Wait Class	SQL ID	Time Wait	Lock Owner	Lock Table	Row Address
+ 2968	15-10-2018 16:04:57	ATLAS_RUCIO_R	ddmusr01	oracle@itrac5237.cern.ch (P003)	aiatlasddm001	User I/O	3j9z45ja2r0ka	0			
− 2631	15-10-2018 15:57:00	ATLAS_PANDABIGMON_R	atlpan	httpd@aipanda108.cern.ch (TNS V1-V3)	aipanda108.cern.ch	Cluster	1q7s4c1gh06g4	0			
4120	15-10-2018 15:55:01	ATLAS_PANDABIGMON_R	atlpan	httpd@aipanda106.cern.ch (TNS V1-V3)	aipanda106.cern.ch		a7gyg8wzabjag	242	ATLAS_PANDA	JEDI_EVENTS_PK	ABN7wTAAeAAotbPAAA
− 3200	15-10-2018 15:40:53	ATLAS_PANDABIGMON_R	atlpan	httpd@aipanda107.cern.ch (TNS V1-V3)	aipanda107.cern.ch	Cluster	5h173q2sd45aq	0			
4353	15-10-2018 15:50:53	ATLAS_PANDABIGMON_R	atlpan	httpd@aipanda108.cern.ch (TNS V1-V3)	aipanda108.cern.ch		5h173q2sd45aq	149	ATLAS_PANDA	JEDI_EVENTS_PK	ABN7wTAAeAAksX5AAA
+ 3891	15-10-2018 14:52:02	ATLAS_RUCIO_R	ddmusr01	oracle@itrac5237.cern.ch (P000)	aiatlasddm001	User I/O	518td3q7yr0vy	1			

Figure 5: Blocking sessions tree view.

- ***SQL statement heat map (Fig. 6)*** - Whenever a SQL statement uses bind variables it keeps the same ID for all executions. Thus, for each SQL ID Oracle stores performance statistics in the Automatic Workload Repository (AWR). Our monitoring provides a heat map that highlights the values increased with more than 20% in comparison with the preceding AWR statistics period.



Percentage increase over the closest previous period

>20% 40% 80% 100% >200%

SQL History Statistics

node	begin_time	plan_hash	module	parsing schema	fetches	sorts	execs	pxexecs	loads	invalid	parse_calls	disk_reads	buffer_gets	direct_writes	rows_proc	cpu_time	elapsed_time	etime_per_exec	iowait	cluster_wait	app_wait	concurrency	plsql_time
1	18-10-18 00:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	7	2	0	0	0	3	227243590	360447605	468323	0	7568.02	18005.78	9002.89	11215.14	406.28	0	0.07	0
1	17-10-18 23:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	0	0	0	0	0	0	23248667	30794358	0	0	897.25	3611.87	1	2855.57	6.78	0	0	0
1	17-10-18 22:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	3	0	0	0	0	0	66131625	87309091	237255	0	1971.18	3636.84	1	1826.05	124.5	0	0	0
1	17-10-18 21:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	0	1	0	0	0	1	34730789	55402659	0	0	1220.83	3602.81	3602.81	2486.57	69.33	0	0	0
1	17-10-18 20:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	0	0	0	0	0	0	16871847	23495692	0	0	728.37	3604.84	1	2992.57	8.65	0	0	0
1	17-10-18 19:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	3	1	0	0	0	1	67859900	105359831	237234	0	2154.8	3564.72	3564.72	1551.79	161.82	0	0	0
1	17-10-18 18:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	0	0	0	0	0	0	40018243	45065386	0	0	1268.55	3612.86	1	2472.99	97.25	0	0	0
1	17-10-18 17:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	0	0	0	0	0	0	61832549	64754796	0	0	1812.15	3611.87	1	1955.59	153.42	0	0	0
1	17-10-18 16:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	3	1	0	0	0	1	43464481	87458610	208518	0	1721.08	3599.74	3599.74	1862.17	290.91	0	0.02	0
1	17-10-18 15:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	0	0	0	0	0	0	15805104	23236852	0	0	736.91	3188.61	1	2514.82	63.46	0	0	0
1	17-10-18 14:00	1798747684	DBMS_SCHEDULER	ATLAS_PANDABIGMON	0	0	0	0	0	0	0	50190288	53264497	0	0	1579.68	3616.82	1	2053.51	263.75	0	0	0

Figure 6: SQL statement heat map.

- ***SQL execution plan(s)*** - The Oracle CBO (Cost-Based Optimizer) might decide to change the execution plan of any query in the database based on consideration of various parameters and data distribution within the tables. DBMon retrieves a list of current and previous execution plan(s), which is useful for understanding why and when a SQL execution plan had been changed.
- ***Storing monitoring data*** - All monitoring data is stored historically, which is extremely beneficial in situations when DB issues occurred outside the working hours.

4 Deployment

ATLAS DBMon uses GitLab CI (Continuous Integration) [18] pipelines for basic testing, containerization and deployment on the CERN Openshift[19] infrastructure. A Docker[20] image of the application is created during the process and deployed via the OpenShift client whenever there is a commit in the Master on GitLab.

The Web-page is protected via CERN SSO (Single Sign-on) and restricted to be accessible only by ATLAS members.

5 Future development

The next step of the ATLAS Database and Application Monitoring development is to replace the Java REST application with the Oracle REST Data Services (ORDS)[21]. This is the Oracle build-in RESTful application which is easy to configure and maintain. Moreover, since the purpose of DBMon is to monitor Oracle databases, another dependency on Oracle technology does not introduce additional problems.

References

[1] ATLAS Experiment: atlas.cern, ATLAS Collaboration, JINST 3, S08003 (2008)
[2] Oracle: oracle.com
[3] RUCIO Project: rucio.cern.ch,
[4] PanDA Project: news.pandawms.org,
[5] WinCC Project: etm.at
[6] ATLAS Database and Application Monitoring: atlas-dbmon.web.cern.ch (available only for ATLAS members)
[7] PL/SQL: oracle.com
[8] Spring Framework: spring.io
[9] Jersey: jersey.github.io
[10] Java: java.com
[11] AngularJS: angularjs.org
[12] Gamma, Erich et al. (1994) Design Patterns
[13] JavaScript: JavaScript
[14] HighCharts Project: highcharts.com
[15] Bootstrap: getbootstrap.com
[16] AngularMaterial: material.angularjs.org
[17] TreeGrid Project: github.com/khan4019/tree-grid-directive
[18] GitLab: gitlab.com, GitLab.
[19] OpenShift: openshift.com
[20] Docker: docker.com
[21] ORDS: oracle.com