The WEB interface for the ATLAS/LCG MySQL Conditions Databases and performance constraints in the visualisation of extensive scientific/technical data

A. Amorim, D. Klose, J. Castilho, L. Pedro, N. Barros, T. Franco, FCUL, Lisbon, Portugal
A. Vaniachine, ANL, Argonne, IL 60439, USA

Abstract

A common LCG architecture for the Conditions Database for the time evolving data enables the possibility to separate the interval-of-validity (IOV) information from the conditions data payload. The two approaches can be beneficial in different cases and separation presents challenges for efficient knowledge discovery, navigation and data visualisation. In our paper we describe the conditions data browser - CondDBrowser - a tool deployed in ATLAS for scientific analysis and visualisation of this data.

A wide availability and access to the overall distributed conditions data repository was achieved through a seamless integration of the IOV and the payload data to the user a unifying web interface that hides the persistency storage details. Another user-friendly feature of the tool is a simplified querying language similar to QBE (Query by Example).

Our case study is based on the web interface developed for the ATLAS/LCG ConditionsDB. The interaction with other payload storage technologies, external to the ConditionsDB, will also be presented. In particular, the integration of the NOVA database technologies.

We will discuss how the information is gathered from the ConditionsDB and the corresponding extensions needed to enable data browsing in the external repositories, how it is organised, and what kind of operations (search and visualisation) are allowed. We'll also present how this interface uses the C++ API extending it to a similar PHP interface, that can be used to browse data collected using any of the ConditionsDB implementations.

Performance constraints are also presented and will be discussed in detail.

DEPLOYMENT

The CondDBrowser was deployed in 2004 to support ATLAS Combined Test-beam data analisys operations. It provides an intuitive interface that enables the end user to inspect data kept in the Conditions Database.

Conditions Database (CondDB) stores data using a temporal extension to the relational model, this means that all data managed by this tool has an Interval of Validity (IOV) associated with it [1]. That data follows an hierarchical organisation, in which tables and blobs are kept inside Folders and FolderSets. The CondDBrowser gives support to this kind of visualisation, and it is now providing navigation and browsing capabilities to more than 8 GB of conditions data in 1800 folders and 6 GB of test data from sub-detectors [2]. This data is growing at a high rate, and the CondDBrowser was developed to scale with the data.

TECHNOLOGY

The CondDBrowser extensively uses the PHP Bind (often called only Bind), a PHP Extension. This module exports the CondDB API to the PHP programming language, enabling the emerging of new tools faced to the World Wide Web development. This is made using a PHP Custom Extension. There are some differences among the API available to PHP and the standard CondDB API, but since we are talking about two programming languages that follow different paradigms, it could not be avoided.

The set of operations implemented by the Bind rely on the Conditions Database. All these operations are read only for two reasons. The first one is because the CondDBrowser does not make write operations, the second is based on security reasons. If we expose access to the Database to the Internet, the best way to avoid corruption of the data (due to malicious or neglected use) is to provide only read only access to the users. Of course this is a limit to the operations available in the Web Interface, and there may be the need to provide write access in the future.

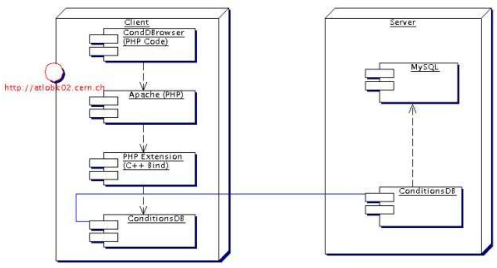

Figure 1: PHP Bind's diagram

As we can see from Figure 1 the Bind does not need to be in the same host as the database. This give the possibility to have several tools running in different hosts operating in the same database. That gives us an obvious advantage when dealing with strong efficiency requisites, since it won't affect the database performance when the system is dealing with HTTP requests.

The Bind only relies on the Conditions Library and this makes it DBMS independent, because the Conditions Library encapsulates the DBMS details. This gives us the

chance to support several DBMS architectures, and if we guarantee that the CondDB API is available to each one of them, then also the Bind will. Also, any changes on the Relational Schema of each implementation won't be reflected to the Bind, that can survive without changes as long as the API stays the same.

Performance issues were also considered, and there was the need to deal with the limitations of the Conditions API. When fetching a bulk of data from the database, there is the need to analyse the data in chunks instead of the several thousands of objects returned by a query. This is a result from the known limitation of the human brain to analyse large amount of information at the same time. Since the Conditions API doesn't has a feature to limit the number of objects returned by the query, the Bind uses a mechanism to return a set of objects from a given interval instead of the full set. The CondDBrowser uses this mechanism intensively, in order to allow the user to analyse one hundred objects at a time. This feature also brings into consideration performance issues. Since the Bind needs to fetch data from C++ and export that data to PHP, we would like to guarantee that only the relevant objects go through this process. Avoiding operations over objects that the user doesn't want to analyse enhances the performance of the system.

Finally we refer to another feature supported by the Bind that is not present in the CondDB API. Usually the IOV is represented by two time stamps, a 'Since time' and a 'Till time'. In the Atlas Experiment it is also usual to see time representations as a 'Since run/event' and 'Till run/event' pair. This differentiation is only made by the PHP bind, and is the only feature not present in the Condition's API. CondDB treats all the IOVs as a pair of Keys, and does not distinguishes between the two representations.

It is good to notice that the complete functionalities of the Condition's API are not exported to the PHP because the PHP Bind was not a milestone. The goal was to put the CondDBrowser up and running, and the Bind is only a way to achieve it.

DATA ANALISYS

ConditionsDB uses features from both Hierarchic and Relational database models. From the Hierarchic model we have Folders and FolderSets, from the Relational model it inherits Blobs and the so called CondDBTables[1]. This is just a brief introduction to ConditionsDB, please refer to[1] for further information.

The CondDBrowser tries to reflect the nature of the Conditions Database, using an intuitive and user friendly interface. It allows the user to browse the hierarchic organisation of Folders and FolderSets, to query Blobs and CondDBTables[1] and to access the payload of NOVA Blobs[2].

[1]CondDBTables also use some features from the Object-Relational model

[2]described in the NOVA Integration section

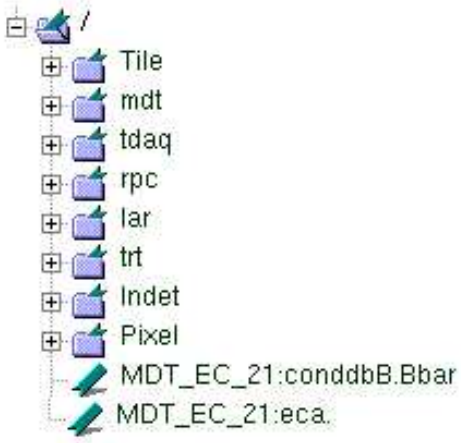

Figure 2: Folder & FolderSets menu

Figure 2 shows the view for browsing the hierarchic organisation of the CondDB available to the user. This menu allows the end user to inspect the FolderSets and select a Folder that contains CondDBTables or Blobs (bot not both). When the user selects a Folder or FolderSet, all the meta information regarding to the item will be available. This information will describe the type of the item (if it is a Folder or a FolderSet). If a Folder is selected, the type of the data stored in it will also be shown (if it stores Blobs, or a CondDBTable). Finally, the description, attributes and path[3] of the item will be available. Figure 3 shows an example of the meta information display.

Path:	**/tile/dcs/System1:CoolingUnit.T0.value:online..value**
Attributes:	none
Type:	Cond. Table
Description:	<addrHeader><address_header service_type="72" clid="94049889" /></addrHeader><typeName>GenericDbTable</typeName><timeStamp>time</timeStamp>

Figure 3: Folder Description

After selecting a target Folder to analyse, the user as the chance to query the data structure contained in it. The data structure may vary between Blobs, CondDBTables, CondDBTables with IDs and CondDBTables with Tags. Although there are significant differences between the various data types, the procedure to query any of them is very similar. A menu is made available with several options and a button, like the one in Figure 4. If any of those options is not available for the chosen data structure, it won't be available and the user will not be able to select it.

The first option (from left to right) regards to the ability to browse a data type with a chosen Tag. Tags are used to take snapshots[4] of the database. Every folder has the Tag 'HEAD' associated to it by default. That Tag retrieves objects[5] with the highest layer. Using this option, the user can browse a data type with the selected Tag. The default Tag is the one named Head, and it retrieves the objects with

[3]the location of the item in the hierarchic model

[4]a snapshot is the exact state of a temporal table at a certain point in time

[5]an object can be seen as a Blob or a Row

the highest layer from the selected data structure.

The second option (when available) gives the user the chance to select all the objects that intercept a given moment in time. Using this feature the user is able to analyse all the versions of a given object, and not only the one with the highest layer. This may be the less used option from all the browsing capabilities of this tool.

Using the last option, the user is able to inspect a group of objects that intercept a given time interval. If layers are available in the data type that the user is browsing, this option will only retrieve the objects with the highest one. In the case where the layer system is not applied[6], the objects returned are all that intercept the given interval.

The combo box available at the lower right corner of the menu is a shortcut. It was made to avoid the fact that the user was always typing the since/till dates for the last N minutes. Using this combo box the user can select minutes in the last hour (with intervals of 5), and use that interval to browse the history of the data.

The small check box located in the bottom of the menu regards to the fact that some IOVs are stored has Run/Event pairs, instead of regular dates. Using this option IOVs are presented to the user in that format. This was a requirement that appeared after the NOVA Integration, described in the next section.

In Figure 5 we can see the result of a query being shown to the user.

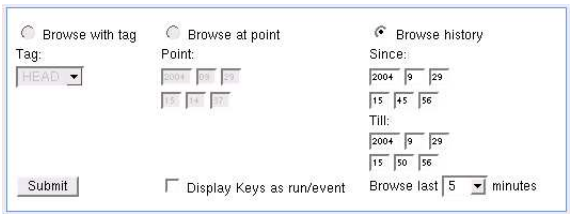

Figure 4: Data inspection menu

Figure 5: Analysing data

The result of a query will be shown in chucks of one hundred objects at a time. This deals with the two situations described before. Performance and the huge amount of data to analyse at a time.

When the user is querying CondDBTables, there is the need to restrict the query to match some values of a given column. For this reason, another menu is also shown to the user when a table is being subject of analisys. The menu shown in Figure 6 is an example on how to make restrictions in the rows retrieved by a query. In this case, the browser will retrieve all rows where the value on the column 'seqnr' is equal to 1. This menu will restrict the results of the query issued with the menu described above. This means that, if the user issues a 'browse history', the CondDBrowser is able to make restrictions in the result. It doesn't mean that the user is able to make a query without taking in consideration the option selected in the menu described above.

The way how the user queries the database was inspired in a simple DML language named 'Query by Example' (QBE)[3]. Although these menus are less expressive than QBE, they provide a very intuitive interface and only requires that the user knows about the Conditions Database philosophy.

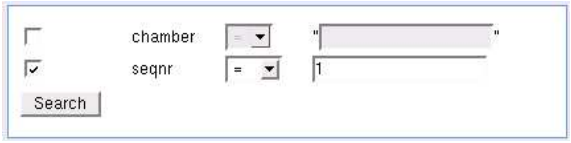

Figure 6: Analysing data

NOVA INTEGRATION

To support conditions data payload storage in NOVA database [4] the CondDBrowser was enhanced with NOVA binary data browsing capabilities. Integration with the NOVA Database could only be achieved with close cooperation between the developers of the Lisbon group and the NOVA developing team. This interaction resulted in a system that detects and interprets NOVA references[7]. After the interpretation of the reference, the browser connects to the NOVA Database and displays the result of the NOVA table.

NOVA Blobs are located inside regular Folders, that can be browsed by the methods described above. After querying one of these Folders, the user will see all the Blobs that match the given query. A Link will appear, allowing the user to inspect the content of the Blob [8]. An example is shown in Figure

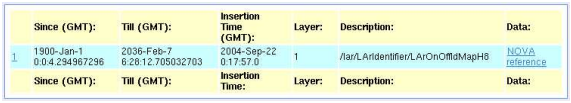

Figure 7: Blob with NOVA Reference

The user may inspect the payload, and after it's done a Link that connects to the NOVA Database will be shown.

[6] i.e. CondDBTables with IDs

[7] NOVA references are kept inside the payload of Blobs

[8] this link is also enabled even if the Blob doesn't contain a NOVA reference, but the name of the link differs

After choosing this Link, the user will be able to browse the NOVA Table and analyse it's contents. Performance was also taken in account in this situation, and the user will see the data in chunks of one hundred rows. In Figure 8 we have an example of a NOVA Table being analysed by a user.

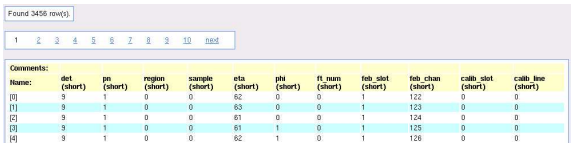

Figure 8: NOVA Table

This was a simple method, and is a great example that teach us how easy it is to achieve interaction between different tools, when several developing teams make some work together. It is very important that developers on one team have the feeling that work is being also made by the developers of the other team. It seems like a logic process, and things work better this way.

FINAL CONSIDERATIONS

After seeing how the CondDBrowser works, it is now time to make one final consideration. This consideration can be seen as a proposal, or even as a challenge since it involves all the developers that create tools that need to interact with temporal databases.

During the development of the CondDBrowser, several discussions where made about the Time Zone used to store/show IOVs. The CondDBrowser assumes that the IOVs stored in the CondDB are in the UTC format. And there are several reasons. The browser can connect to any database, this includes any time zone possible. Also, the user may be using the browser in a time zone different from the one used by the database. But the most important reason is that the database may be distributed, and this gives the chance to have several clusters in different time zones.

Also remember that in some countries have summer and winter time. If some data is stored in winter time and the user analyses it during summer, he needs to have the conversion in mind. This conversion can also be coded in the tools, but that would generate a lot of confusion.

For all these reasons, our proposal is to store IOVs in the UTC format. It is hard to believe that other solutions may be reasonable than this one.

ACKNOWLEDGMENTS

We thank all of our ATLAS collaborators and, in particular, the developers from the online and offline software domains.

Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, under contract W-31-109-Eng-38.

REFERENCES

[1] A. Amorim, J. Lima, L. Pedro, D. Klose, C. Oliveira, N. Barros. IEEE-NPSS: An Implementation for the ATLAS Conditions Data Management Based on Relational DBMSs. In Proceedings of the 13th IEEE-NPSS Real Time Conference, pp. 591- 595, May 2003.

[2] S. Schmidt conditions_ctb_2004 Folder Summary/Folder Details. Available: http://www.abstracture.de/atlas/ctb.html

[3] Henry Korth, Abraham Silberschatz, Database System Concepts, USA: McGraw-Hill Education, 1997

[4] A. Vaniachine et al, CHEP'03, La Jolla, USA, 2003, eConf C030324, MOKT006 (2003) [cs.db/0306103]