RELATIONAL DATABASE IMPLEMENTATION AND USAGE IN STAR

M. DePhillips, J. Lauret, V. Perevoztchikov, BNL Upton, USA
J. Porter, University of Washington, Seattle WA, USA

Abstract

The STAR experiment at Brookhaven National Laboratory's Relativistic Heavy-Ion Collider has been accumulating 100's of millions events over its already 5 years running program. Within a growing Physics demand for statistics, STAR has more than doubled the events taken each year and is planning to increase its capability by an order of magnitude to reach billion event capabilities by 2008. Under such a rate stress imposed by the event rate, the run condition support and database back-end needed to rapidly mature to follow the demand while preserving user convenience and time evolution but also allow for in depth technology as required.

This document presents the use of relational databases in STAR organized as a three tier architecture model: a front-end user interface, a middle tier home grown C++ library (StarAPI) that handles all of the unique requirements arising from an active experiment, and finally, the lower level DBMS requirements and data storage. Paramount considerations include maintaining flexibility and scalability with modular construction and consistent namespace; ensuring long-term analysis integrity with three-dimensional time stamping or range of validity which in turn allows for solid schema evolution; and ensuring uniqueness with expanded primary keys. This paper identifies and discusses trade-offs and challenges that have occurred during the evolution of the STAR experiment, and specifically the challenge introduced by detectors which could only be described in terms of million leaves within an ultra-fine granularity of calibration values.

INTRODUCTION

There is often a gulf between principle and practice when designing database systems, especially those that are to be used in a dynamic experimental setting. These differences arise as new, unexpected requirements are discovered and implemented. Also, data volume tends to increase in unforeseen ways as technologies improve over time. Careful attention to the design phase of these data base systems can provide a scalable and flexible system that can adapt and absorb the changes while providing a consistent interface to the end user of the system.

The STAR database system has proven to date, that its initial design is robust enough to grow within the demands of an active experiment on the Relativistic Heavy Ion Collider.

Background

The following decisions resulted from the design phase of STAR database system

- That the online and offline database system are inherently different in their purpose and their operations therefore should remain separate from each other.
- MySQL would be used as the main DBMS for both systems
- For the Offline system, an in-house API library should be build using well defined requirements

.

This system was deployed in-full in 2000. Modifications have been made to adapt to the evolving demands, however, the user interface layer has remained constant through the years.

CURRENT SYSTEM

Online

STAR's online database system is primarily a data-taking system. These data include condition and monitoring data from the detectors and the collider and, event, file and run tag information from the data acquisition system. The exception to this writing is an aggregate series of tables that compile vital run information and display this information in real time on the web.

The detector and collider data are streamed in to the database tables using a suite of C++ based daemons that poll STARs slow controls system (i.e., EPICs). Each daemon is specific to one data-source (e.g., a specific detector) and writes to a specific database. Due to the specificity of the daemons and since they are compiled C++ code they are fast, have a small footprint and are independent of each other, thus creating no dependencies.

The Run, File and Event tags are written directly into the databases, using the MySQL C API, from the data acquisition. By far, the largest database in the STAR system is the Event Tag which contains information about each event taken in the various detectors. Table 1 shows the growth of events taken over years of RICH running. It is with this table that the system faces it's only stress, primarily due to its size. MySQL handles these issues nicely by providing a fast storage engine and the ability to compress and remerge tables which allows for the

"rolling over" of tables, keeping them at a manageable size.

The choice to use MySQL for this system was initially based on its flexibility and its writing speed. MySQL allows for many databases, containing many tables, to exist on one instance of its server. This allows for a clean division in data storage. Its MyISAM storage engine is a suite of three files that require no expanded table space typical to databases that support transactions. With no transactions needed, just in direct simple writes, this engine is ideal for fast data collection. STAR has been steadily writing at an average of 200 HZ into these tables without a problem. It should be noted that there is a one field integer index on the largest of these table, which are allowed to grow to roughly 20 million records before they are rolled over.

Table 1: Online Data Stored in EventTag DB

Year	**Events (Millions)**	**Storage Size**
2001	3	21 MB
2002	51	50 MB
2003	148	98 MB
2004	300	68 GB
2005	568	110 GB

Online Topography

The online system contains one linux node with three instance of MySQL running. These configurations allows for access to the three discrete sets of databases through three separate ports. This helps with bandwidth. The data acquisition tables are separated from the conditions databases. Both these port aggregate data to a third port which contain the Run Log tables. These tables support the real time monitoring of the runs and require reads from PHP code access via the WWW.

There are two backup nodes, one that simultaneously collects conditions data and receives periodic back-ups of data acquisition data. A second node receives nightly dumps of all conditions and run log tables.

Online to Offline

Much of the online conditions data is used in offline analysis as calibrations values. Latency is not an issue here since offline analysis is not normally done in real time. For completeness, it is worth mentioning that there is a "fast offline" system in place that analyzes a sample of recently received data, however, its tax on the database system is negligible, therefore, not discussed in detailed here.

The migration of these values is done with a series of ROOT marcos triggered by cron at various intervals ranging from every five minutes to once an hour. These macros are similar to the online daemons in that they map a specific data source to a specific table. They differ in that they do not strictly stream data. The macros take averages, intervals and make decisions as to what values are appropriate for offline analysis.

Offline

The star offline system serves primarily the reconstruction and analysis of data. The data base system consists of suite of discretely organized databases each containing a set of storage tables and a home-grown API interface from user to tables. The following design considerations where taken into account when composing the API which in term led to the design of the tables and the databases that contain them.

- SQL should be hidden from user code
- Monitor Loads and Economize usage
- Independence for all other frameworks
- Data is access using generalized code.

The API accomplishes these goals by providing uses with at set of generalized methods for retrieve and writing data, building hierarchies and trees for data retrieval. Also the API uses four API specific tables that sit alongside the actual data storage table to pre-define indexes, maintain knowledge of it own schema and provide transparent grouping mechanisms. The grouping and indexing is accomplished, in conjunction with these tables, with a five element primary key.

Offline-API

The API relates to the database storage tables via a set of naming conventions that is used in place of a catalogue of available data sources. These naming conventions are of the following format: `Domain_type`. Domain represents the usage of data that the database contains, for example `calibration`, `condition`, or geometry data. Type, represents the subsystem or detector `the` data belongs to, for example, `tpc`, `ssd`, or `emc`. Therefore, an example of a database name would be `Calibrations_tpc` or `Geometry_ssd`. This convention is very useful in creating trees for retrieval of data during production or analysis. For example, the domain is a parent leaf to children types which are in turn parents to their tables etc… This type of tree is copied into memory once and standard tree parsing algorithms are use to traverse the trees in search of data.

The indexing, a grouping of retrieved data, is based on a five element primary key. The key consists of

- `beginTime` – A time stamp that denotes the beginning of a validity period – usually associated with the beginning of a star data taking period, confusingly also called a run, "STAR Run". (the running period of the yearly active collider will be referred to as a "RHIC Run".)
- `endTime` - which denotes the end the time of a table/detector. It is important to note this is different than the end of a valid period such as a STAR run.
- `Flavor` – a grouping flag which denotes a particular type of data e.g., "simu" for simulation data, or "ofl" for offline data.
- `ElementID` – a finer grained grouping mechanism that allows for multiple rows to be returned with one time stamp.
- `entryTime` – real time entry of data values

The STAR API timestamp is three dimensional, containing the `beginTime` and `entryTime` from the primary key and another time field called `deactive`. This field, when set as a time value, turns a record off from that time forward. This timestamp scheme creates validity time ranges: one `beginTime` to the next `beginTime` allows for corrections to be made by deactivating a record and allows for schema preservation by the use of the `entryTime`. When code and database values are frozen for a production, a production time is announced. The production time creates a snapshot in time which secures the ability for a user to go back in time and recreate a production. This is true even if a value has been deactivated and changed at a later date.

The API also uses a table that contains a self description of the schema of the database. The purpose of this is to provide independence from any external framework that interacts with the API. Much like a web service, the API connects to the outside world by accepting generalized fetch of write requests and passes back a shapeless object `(void *)` with a self-descriptor. It is these self descriptions that are stored in the aforementioned table.

In terms of the actual library the STAR DB API is itself modelled after three tier architecture. The High level user interface remains static; no changes have been made to those classes/methods since the inception of the API. This consistency increases user expertise and acceptability of the system. The other tiers contain STAR specific wrappers around low level SQL and operations and management methods.

Offline Topography

STAR Offline database system uses MySQL replication to set up an immediate distribution of database value for the purpose of load sharing both locally at BNL and globally. At BNL there is one master and two pools of slaves. Three nodes are dedicated to analysis and have conduits through the BNL firewall to the outside world. Four nodes are for internal reconstruction. These machines support a farm of 545 nodes. There are an additional 5 nodes which connect to the BNL master as slaves that are distributed to STAR global community of collaborators. Each node functions well up to about 300 concurrent connections, at which the machine responsiveness depletes. The load sharing as is, keeps the maximum connection to each node at about 150 threads.

ONGOING DEVELOPMENT

The very nature of an active experiment is dynamic and as mentioned in the previous section one of the goals of the STAR database system is to maintain a consistent generalized interface for users. This means all alterations and adaptations must take place either on a table-schematic level or in the lower level modules of the API. To date, all new requirements of the databases system has been able to be absorbed by these levels. Below is a case study that represents a typical transition that the database system must support.

Silicon Strip Detector Calibrations

With the commissioning of the 6th RHIC run STAR began to integrate its Silicon Strip Detector (SSD). The SSD provided some unique challenges to the database system in that it contains 491520 points of calibrations. Further, any number of these calibrations could be tweaked on a STAR Run by Run basis. The traditional method for recording Calibration tweaks would be to provide a new block of data, grouped on the `elementID` delimited by the `beginTime`. In this case each block would contain ½ million records. This, of course, presents both storage and performance issues.

Specifically the following three issues needed to be addressed.

- Size of returned data set
- Performance hit based on large queries
- Size of the storage table

The respective solutions are as follows:

- Data Packing
- Query Optimization
- Schematic Architecture of tables

The size of each row of data was reduced by a factor of two by packing integers into chars when it assured that the value will not be greater than 255. In the case of the SSD, two out of three fields could be stored as chars, or "tiny ints" in MySQL terminology. Bit masking is also a useful technique to pack data into smaller containers.

The low level SQL contained the `where` clause operator "`in`". This is a good generalization technique

that tightens code, in that it accepts both many and one parameter, therefore no decision has to be made. However with the arrival of the SSD the '`in`' clause contains 491520 parameters. By adding some decision making logic and introducing the "`between`" operator, the API improved performance by greater than a factor of two for queries that included the SSD and thus, is able to handle large amount of data paradigm.

In the original database system the size of the storage table increase by the indexed block of data each time a value is change. In the case of the SSD, that would mean an additional 491520 rows with each tweak. This is beyond reason and the API and schema needs to be changed to accept only the changed values and bring the unchanged values up in time while also leaving them available to their original timestamp. This work is ongoing.

Online API

As STAR continues there is a greater emphasis to quickly analyze data in from the online systems. This means unforeseen requests to the data tables. Unlike, offline, however, requests come with greater urgency and variety of ad-hoc methods are used to access data from the database. New ideas stabilize without any attempt at standardization. Different users prefer different tools creating a mix of technology. Administration of the servers load becomes difficult. These issues point directly at the need for an Online-API that mimics the functionality of the STAR offline API.

The original STAR database design still holds for the development of the online API. For example the three tier architecture would remain, the same naming conventions could be used, and most classes would not need to be changed. The main differences would be in grouping methods. Based on present usage, online grouping comes more in terms of by run or by trigger as opposed to by timestamp in offline. This is an ongoing project.

CONCLUSIONS

The STAR database system has proven to be a scalable and flexible, by providing users with a stable consistent interface to database values while handling the rigors of a production environment of an active experiment.

The system continues to evolve with this established framework to handle new challenges that develop as the experiment matures.

REFERENCES

[1] MySQL http://www.mysql.com/

[2] STAR Database documentation http://www.star.bnl.gov/STAR/comp/db/

[3] Porter R. J. "STAR Database Implementations with MySQL". Proc. Of CHEP 2001 CERN 2001-2-030, 2001.