# Petaminer: using ROOT for efficient data storage in MySQL database

**J Cranshaw[1], D Malon[1], A Vaniachine[1], V Fine[2], J Lauret[2] and P Hamill[3,4]**

[1]Argonne National Laboratory, 9700 S. Cass Ave, Argonne, IL, 60439, USA

[2]Physics Department, Brookhaven National Laboratory, Upton, NY 11973 USA

[3]Tech-X Corporation, 5621 Arapahoe Ave, Suite A, Boulder, CO 80303, USA

[4]paulh@txcorp.com

**Abstract**. High Energy and Nuclear Physics (HENP) experiments store Petabytes of event data and Terabytes of calibration data in ROOT files. The Petaminer project is developing a custom MySQL storage engine to enable the MySQL query processor to directly access experimental data stored in ROOT files. Our project is addressing the problem of efficient navigation to PetaBytes of HENP experimental data described with event-level TAG metadata, which is required by data intensive physics communities such as the LHC and RHIC experiments. Physicists need to be able to compose a metadata query and rapidly retrieve the set of matching events, where improved efficiency will facilitate the discovery process by permitting rapid iterations of data evaluation and retrieval. Our custom MySQL storage engine enables the MySQL query processor to directly access TAG data stored in ROOT TTrees. As ROOT TTrees are column-oriented, reading them directly provides improved performance over traditional row-oriented TAG databases. Leveraging the flexible and powerful SQL query language to access data stored in ROOT TTrees, the Petaminer approach enables rich MySQL index-building capabilities for further performance optimization.

## 1. Introduction

The Petaminer project [5] is part of the effort to provide physicists with efficient tools for query and analysis of very large-scale data generated by High Energy and Nuclear Physics (HENP) experiments, for example ATLAS at the Large Hadron Collider (LHC). Such experiments store Petabytes of data in ROOT files described with TAG metadata and have challenging goals for access to this data. Physicists need to be able to compose a metadata query and rapidly retrieve the set of matching events. These skimming operations will be the first step in data analysis. Improved efficiency will facilitate the discovery process by permitting rapid iterations of data evaluation and retrieval.

To address this problem, we have prototyped a custom MySQL storage engine to enable the MySQL query processor to directly access TAG data stored in ROOT TTrees. In addition to the efficient SQL query interface to the data stored in ROOT TTrees, the Petaminer technology links MySQL index-building capabilities to FastBit bitmap indexing of the data in ROOT TTrees, providing further optimization to TAG query performance. A preliminary report of this work was made at the

ACAT 2008 workshop [6]. This paper reports specific performance results we achieved by using FastBit indexing to optimize reads of data stored in ROOT.

## 2. Approach

Petaminer integrates several middleware components into a custom MySQL database engine. MySQL [7], ROOT [8], and FastBit [9] are standard tools. The custom code developed for this project resides in the Petaminer MySQL storage engine. MySQL's open storage engine plugin architecture makes writing a custom engine straightforward. Users interact with the MySQL engine via standard SQL APIs using command-line clients or other tools compatible with MySQL. Figure 1 shows the high-level Petaminer architecture.

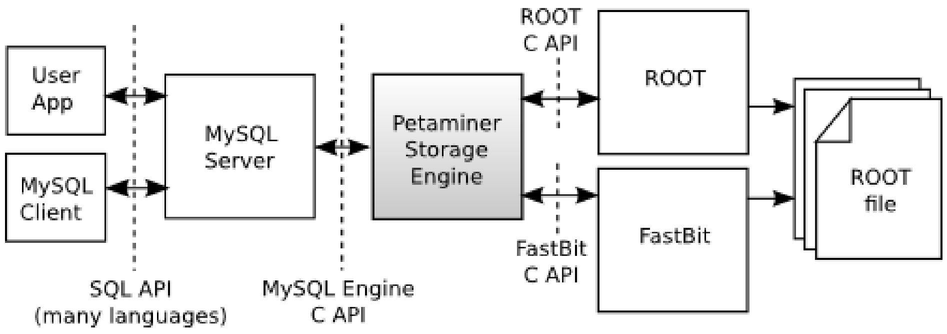
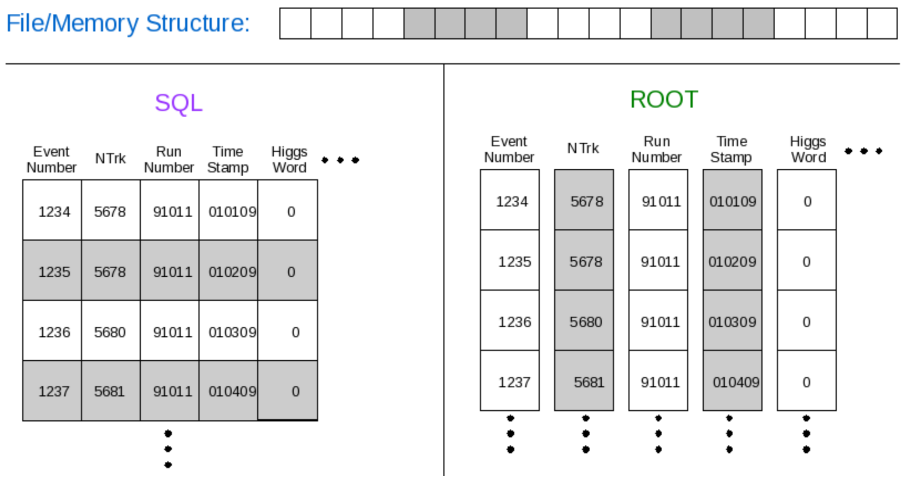


**Figure 1**. The Petaminer architecture.

The most important advantage of this approach is that data remains in the ROOT files rather than necessitating a bulk copy from ROOT into standard database tables. Since the TAG metadata schema evolves over time, using conventional database tables would require corresponding schema maintenance, and copied data would have to be periodically reloaded, a major operational burden for experiments producing very large volumes of data.

Petaminer also leverages the advantages of data storage in ROOT, which was designed for large scale physics data. ROOT stores TAG metadata in column-oriented TTrees, which are more efficient for read-only queries than a conventional database's row-oriented storage. Figure 2 compares conventional database row-oriented data structure and ROOT's column oriented format.



| Event Number | NTrk | Run Number | Time Stamp | Higgs Word | • • • |
|---|---|---|---|---|---|
| 1234 | 5678 | 91011 | 010109 | 0 | |
| 1235 | 5678 | 91011 | 010209 | 0 | |
| 1236 | 5680 | 91011 | 010309 | 0 | |
| 1237 | 5681 | 91011 | 010409 | 0 | |

| Event Number | NTrk | Run Number | Time Stamp | Higgs Word | • • • |
|---|---|---|---|---|---|
| 1234 | 5678 | 91011 | 010109 | 0 | |
| 1235 | 5678 | 91011 | 010209 | 0 | |
| 1236 | 5680 | 91011 | 010309 | 0 | |
| 1237 | 5681 | 91011 | 010409 | 0 | |

**Figure 2**. Row-oriented database format versus ROOT column-oriented data structure.

Conventional databases such as MySQL normally store table data as rows, which is advantageous for row-based writes, updates and transactions. However, querying a column can require accessing data across the entire table since all values are distributed across all rows. A column-oriented data structure is more efficient for read access since all values of an attribute are stored continuously. Also, column orientation permits efficient, lossless run-length encoding [10] compression of values, which can greatly reduce the storage size (and thus the read time) of fields containing a limited set of discrete values, as is common in TAG metadata. Since most physicists perform reads much more often than writes, the read performance advantage of column-orientation is desirable. Furthermore, ROOT files can be distributed across storage devices, so do not have the size limitations of regular database tables. Reads across ROOT files via XROOTD [11,12] or a similar distributed storage protocol can potentially be parallelized using multiple processes, offering further potential for performance improvement.

Making the ROOT data accessible via MySQL also improves flexibility by adding SQL syntax and standard database tools to ROOT data access mechanisms which currently use a low-level C++ interpreter.

By creating a custom storage engine and a C wrapper to ROOT's C++ API, we implemented middleware that maps SQL constructs such as tables and columns to ROOT structures such as TTrees and attributes. Table 1 below gives a simple example of this mapping.

**Table 1**. Parallel ROOT and MySQL Constructs

| ROOT | MySQL |
|---|---|
| TFile **MyData**.root {<br>TTree tree {<br>    TBranch<Int_t> **a1**<br>    TBranch<Float_t> **a2**<br>    TBranch<Char_t> **a3**<br>  }<br>} | TABLE **MyData** {<br>  **a1** INT,<br>  **a2** DOUBLE,<br>  **a3** VARCHAR<br>} |

This automated mapping permits standard SQL statements such as:

```
SELECT a1, a2 FROM MyData WHERE a1 > 100;
```

to map to corresponding ROOT selection and read operations and return the matching values as a SQL result set. Many MySQL compatible tools support performing such SQL operations, including command-line clients, spreadsheets, data mining applications, and Web-based GUIs. The MySQL API can also be utilized in virtually every popular programming language. Thus, this functionality maximizes the ease of access to data stored in ROOT without requiring an application to be tied to ROOT itself.

Once this basic functionality was in place, we turned to improving the performance of queries using the Petaminer engine. Here, we built upon previous ROOT-FastBit work [13] to create middleware enabling the FastBit indexer to create and use indices of ROOT file contents. Petaminer maps MySQL indexing-related operations to ROOT-FastBit calls. This SQL command to create a table with an index named index1:

```
CREATE TABLE MyData (a1 INT, INDEX index1(a1));
```

is translated by the Petaminer storage engine into the ROOT-FastBit calls:

```
TFile* file = new TFile("MyData.root");
TTree *tree = (TTree*) file->FindObject("tree");
TBitmapIndex index;
index.Init();
char indexLocation[16] = "data/test";
index.ReadRootWriteIndexFile(tree, indexLocation);
index.BuildIndex(tree, "a1", indexLocation);
```

When a MySQL query is performed against the table, the FastBit index is used to retrieve the set of values matching the WHERE clause. Depending on the data distribution and WHERE clause parameters, this can be much more efficient than a non-indexed read, which scans all entries in the file.

## 3. Results

Performance benchmarking for FastBit-indexed ROOT data was undertaken by generating data containing randomly distributed numerical values, in order to avoid a non-uniform data distribution that could affect indexing performance either favorably or unfavorably, as well as many columns of different data types that are representative of TAG metadata in HENP production data. Identical data was used in tables read by the Petaminer engine and the standard MySQL engine. The data was indexed using both FastBit and standard MySQL indexing. Queries to select a subset of the values were performed repetitively to generate average timings. The specific query timings here were generated with a standard MySQL configuration on x86_64 Linux (Fedora 8, 2x2.66 GHz CPU, 3 GB RAM). Similar comparative timings are seen on different systems. Figure 3 below shows average query times using the Petaminer and MySQL engines with TAG metadata sizes from $10^5$ to $10^8$ entries.

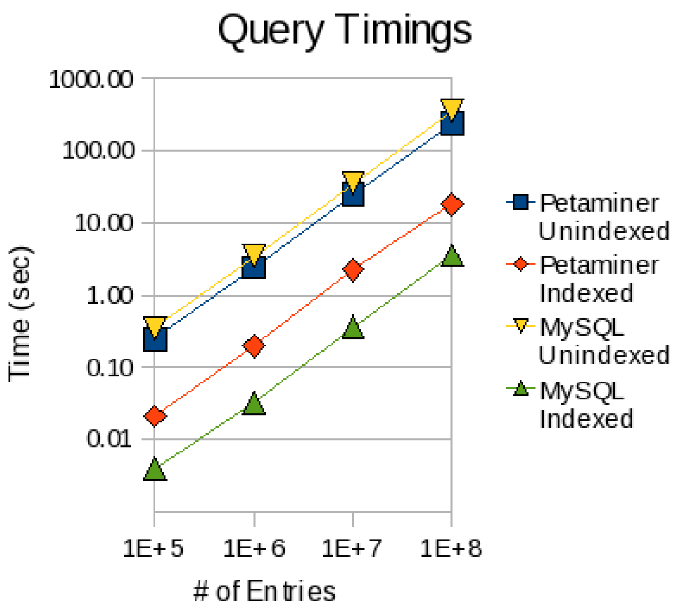


**Figure 3**. Query timings for Petaminer and MySQL engines (log-log scale). *Petaminer Indexed* queries were indexed using FastBit. *MySQL Indexed* queries used standard MySQL indexing.

## 4. Conclusions

Non-indexed queries performed using the Petaminer engine were about 40% faster than non-indexed queries on the same data stored in conventional MySQL tables. This confirms the hypothesis that ROOT's column-oriented data organization offers advantages in read performance over conventional row-oriented database storage.

ROOT reads indexed by FastBit were about 10x faster than non-indexed ROOT reads. This is consistent with previously published ROOT-FastBit performance results, and suggests that the Petaminer interface to FastBit does not incur a significant performance penalty.

Conventional MySQL indexed queries on the randomly distributed data performed 5-6x faster than FastBit-indexed queries, which demonstrates the value of Petaminer's capability to enable choosing an indexing strategy that is best suited for scientific data mining. This also suggests that more effort can be made to optimize Petaminer's FastBit-based indexing functionality. Related work suggests that bitmap indexing can be highly performant in similar HENP analysis scenarios [14].

## References

5 Petaminer project page, https://ice.txcorp.com/trac/ petaminer
6 P. Hamill, J. Cranshaw, D. Malon and A. Vaniachine, *Petaminer: Efficient Navigation to Petascale Data Using Event-Level Metadata*, Argonne National Laboratory Preprint ANL-HEP-CP-09-5, January 26, 2009, to appear in the *Proceedings of the XII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2008)*, Erice, Italy, November 3-7, 2008
7 MySQL main page, http://www.mysql.com
8 R. Brun, F. Rademakers, *ROOT - An Object Oriented Data Analysis Framework*, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch.
9 Kesheng Wu. *FastBit: an efficient indexing technology for accelerating data-intensive science*, J. Phys.: Conf. Ser. 16, pp 556-560.
10 Run-length encoding http://en.wikipedia.org/wiki/Run-length_encoding
11 A. Hanushevsky and A. Dorigo and F. Furano, *The Next Generation Root File Server*, Proc. CHEP 2004, 2004.
12 XROOTD software is available at http://xrootd.slac.stanford.edu
13 K. Stockinger, K. Wu, R. Brun, P. Canal, *Bitmap Indices for Fast End-User Physics Analysis in ROOT*, Lawrence Berkeley National Laboratory Paper LBNL-58426, July 26 2005.
14 K. Wu, J. Gu, J. Lauret, A. M. Poskanzer, A. Shoshani, A. Sim, and W. Zhang. *Grid Collector: Facilitating Efficient Selective Access from Data Grids*. In Proceedings of International Supercomputer Conference 2005, Heidelberg, Germany.