Report on the Reference Database for CMS Monte Carlo Production RefDB

The ARDA Group

Editors: J. Herrala, J. Andreeva

Description of RefDB

RefDB is the CMS Monte Carlo Reference Database [REFDB]. It was used for the first time during the Spring 2002 CMS Computing Production. RefDB is also used by the CMS Grid prototypes in both the USA and Europe to allow them to produce data for the CMS physics community.

RefDB has four functionalities:

- RefDB serves as a production request submission system for the physicists. Web forms allow physicists to define all the request details, including every parameter of physics software. RefDB records the production requests done by the physicists.
- The production process needs to distribute the work to the regional centres and to trace the production progress. RefDB can be accessed via a http client, eg. web browser, by worklow planner tools, eg. McRunjob, as the central source of production instructions. This mechanism allows a high level of automation in the production process.
- RefDB is used for the coordination and monitoring of the world-wide distributed production. Jobs are monitored locally, eg. using BOSS, and metadata is transferred asynchroniously to RefDB for validation and book-keeping.
- Finally, RefDB is a metadata catalogue for the physicists who want to find out what data is available and how to access it.

RefDB is based on a MySQL database designed for CMS Monte Carlo Production book keeping. The (user) access to RefDB is mostly done with php scripts via the CMS web server (cmsdoc.cern.ch), which is an Apache web server running php extensions. Users must respectively use the http getter method, eg. wget command, in order to fetch the data.

Measuring the performance

The goal of the this study is to get quantitative experience on the performance of RefDB in a physics analysis environment. The main objective is to find out how many concurrent connections/requests RefDB is capable to handle robustly and in an acceptable time.

The implemented queries were similar to ones used for extracting from RefDB information for the creation of the set of production or analysis jobs for a given data collection. Information extracted from RefDB represents the table, every row contains the set of parameters required for creating of a given job:

1. Run number (integer).
2. XML POOL catalog fragment in a zipped form (txt). XML POOL catalog fragment is an

extraction of full record of XML POOL catalog containing references to the input files corresponding to a given run number, it does not keep trace of a real physical file location , but has full information about logical file names and META file attributes used by COBRA application. Catalog fragments are used for recreation of the full POOL XML catalog corresponding to a given data collection and used for the further analysis.

3. So called RUNID value, which is required by COBRA application and is used as a key value while attaching data files to the META files of a given collection (txt). Several data collections of different sizes (different number of records in the corresponding table) have been tried.

The performance of the data transfer was analysed by using data collections of different sizes (different number of records in the corresponding table) ranging from 1Mb to 20Mb of data:

Number of records in a collection	Size of a record	Size of the whole collection
209	5.55 kb	1.2 Mb
985	5.55 kb	5.5 Mb
3487	5.55 kb	19.5 Mb

Table 1: The collection size listing in the performance analysis.

First round of stress testing was done in a read-only mode. To simulate the access to data store during data taking, we updated RefDB every 10 seconds with information which a common production job sends to data base when it is successfully finished. However, this did not have a measurable impact on the performance.

Performance tests

One should note that the performance tests were run on the RefDB development server, 147MHz Ultrasparc 5 workstation, which today can be regarded as an low-performance machine. However, the test suite produced by this study can easily be applied on the production database as well. The Apache/php front-end is run on cmsdoc, which is a Sun Sparc Ultra 60. The client is a P4-machine with 512MB of memory. All the machines are connected to a fast ethernet of 100Mbit/s. In order to simulate concurrent user requests, the client machine is running a Ruby script which forks several processes which simultaneously send requests to the RefDB server via the Apache/php front-end. The time-to-complete of these requests was measured with an increasing amount of concurrent processes. In addition the size of the requested collections was varied, where the size of the collection equals the number of records selected from the SQL database back-end (as in Table 1).

The result is shown in Figure 1. For large requests or high numbers of clients the clients failed on the request. A limiting factor was the amount of concurrent connections that the database can handle. A typical error message from the server is:

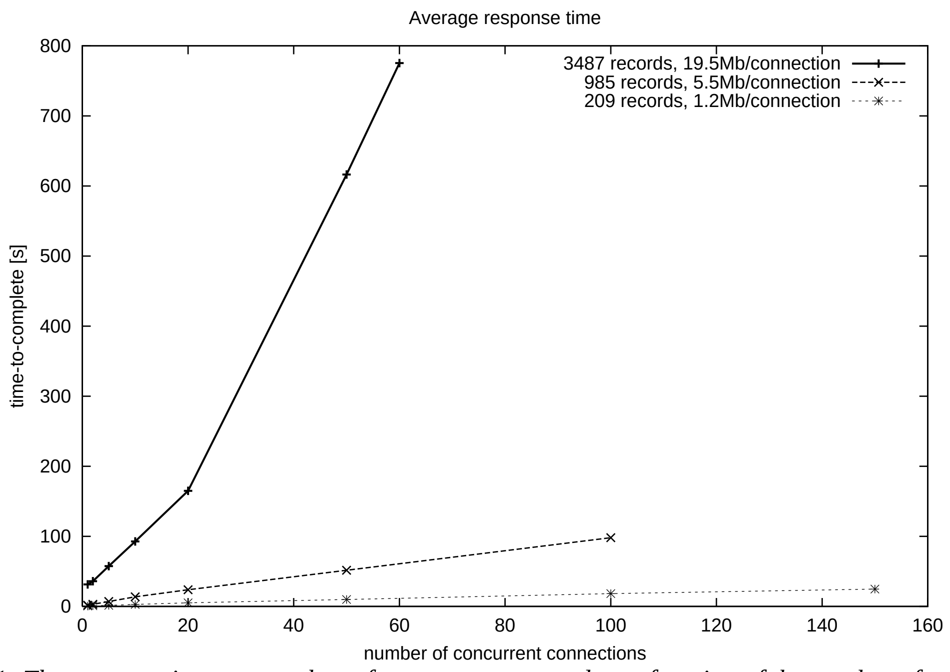

Figure 1: The average time-to-complete of a request presented as a function of the number of concurrent queries. The collections were discribed in the Table 1.

```
Warning: mysql_connect(): Too many connections in
/afs/cern.ch/cms/production/www/RefDB_test/cgi_dev/myadmin/lib.inc.php on line 311
Error

MySQL said: Too many connections
```

This error state prevents the system from being overloaded. Actually, the system was acting in a robust manner during the whole test chain. No crashes were recorded.

Figure 2 shows at which rate the data is transferred from RefDB to the clients. It can be seen that the trasfer rate saturates as a function of concurrent connections. The maximum value depends on the size of a single request that is the amount of records in the collection. It was observed that the most probable bottleneck of the overall performance is the CPU limitation of the MySQL server, which uses about 100% of the available CPU by dealing only with 2-5 concurrent requests.

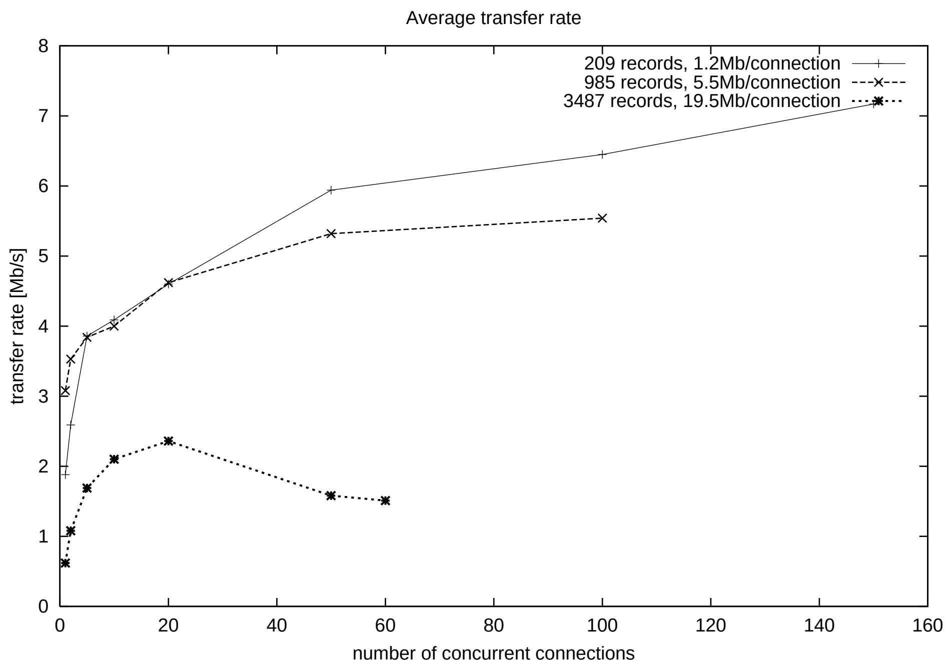

Figure 2: Average data transfer rate between RefDB and clients as a function of the number of clients.

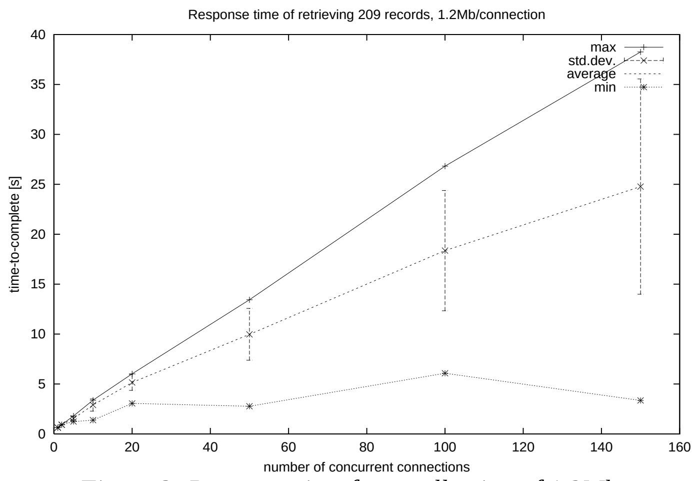

Figure 3: Response time for a collection of 1.2Mb.

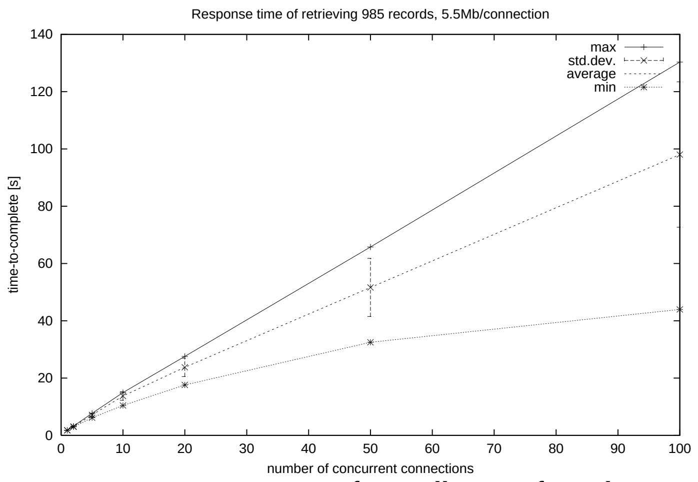

Figure 4: Response time for a collection of 5.5Mb.

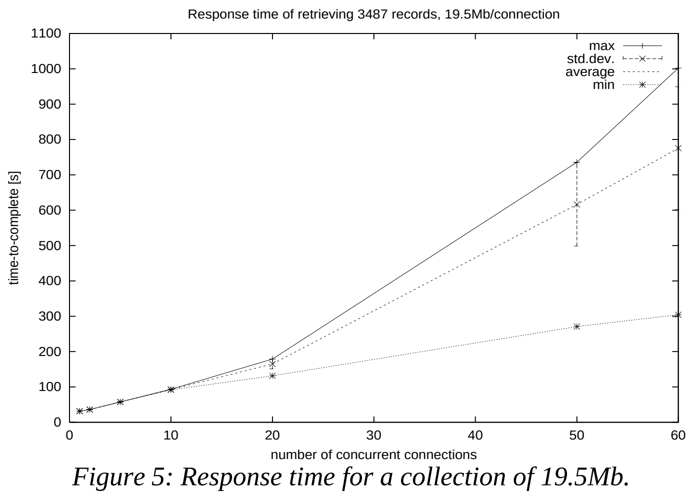

Figure 5: Response time for a collection of 19.5Mb.

Figures 3 to 5 present the response time of RefDB as a function of amount of concurrent connections. It should be noted that the bars present standard deviation, not measurement error.

Figures 6 to 8 present the distribution of time-to-complete for individual queries. The distributions presents the largest successful amount of parallel connections for each collection.

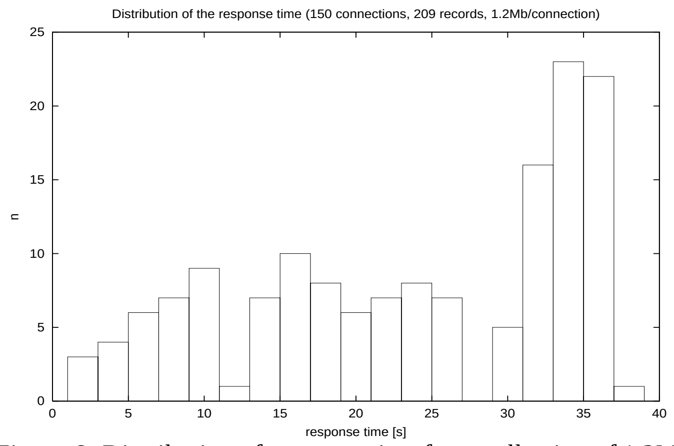

Figure 6: Distribution of response time for a collection of 1.2Mb.

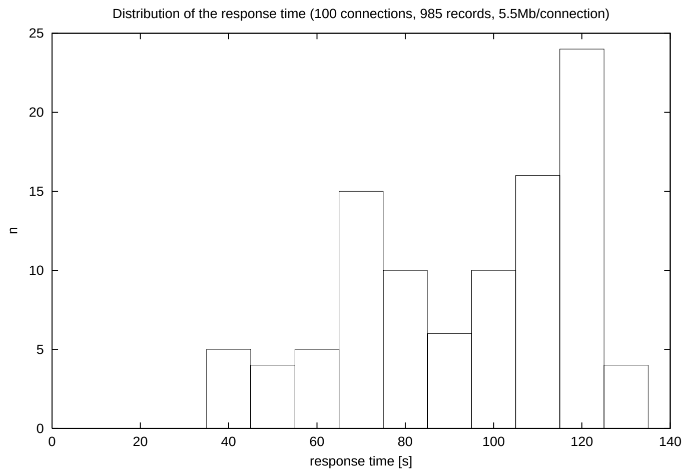

Figure 7: Distribution of response time for a collection of 5.5Mb.

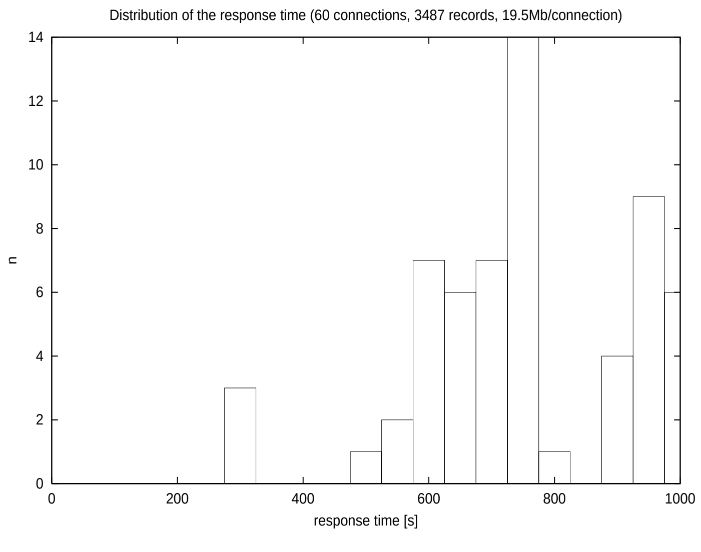

Figure 8: Distribution of response time for a collection of 19.5Mb.

Conclusions

The response times were measured for the RefDB development server, which has an order of magnitude slower processor that the production server. Nevertheless, the results of the stress test were acceptable in current state. There should not be need to serve more
that 50 users concurrently, which RefDB is able to handle even with large collections of 20 Mb. Another aspect is the response time, which varied from 10 to 600 seconds for 50 concurrent users depending on the size of the collection.

The implemented queries were similar to what is done while extracting from RefDB information for creating of set of production or analysis jobs for a given collection (run numbers, catalog fragments to construct an XML POOL catalog, information used by COBRA application).

Bibliography

REFDB: Lefebure V., Andreeva J., RefDB: The Reference Database for CMS Monte Carlo Production, 2003