LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Note **LIGO-T000089-00 - E** 8/30/00

Plan for the LDAS Database Mock Data Challenge

Peter Shawhan

Distribution of this draft:

LDAS; LIGO/LSC Software Committee

This is an internal working note
of the LIGO Project

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS 20B-145
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

WWW: http://www.ligo.caltech.edu/

Contents

1 OVERVIEW

This document describes the plan for a “Mock Data Challenge” (MDC) focusing on the relational database which is part of the LIGO Data Analysis System (LDAS). This MDC has several distinct goals:

- Validate the database table design
- Test the functionality and performance of the hardware and software components of the core database system
- Test the process of inserting triggers and other data generated by the Data Monitoring Tool (DMT) system into the database
- Evaluate user interface tools (both graphical and program-based)
- Exercise various database administration tasks
- Ensure that accurate documentation exists for all aspects of the system.

The general plan is to prepare software, procedures and documentation during the months of Sept.-Nov. 2000 and then to schedule an intensive test period (the MDC itself) in late November. It is likely that some aspects of the MDC will be performed at the observatories (for example, data insertion performance tests) while others will be performed at Caltech (for example, database administration tasks).

2 HARDWARE AND SOFTWARE COMPONENTS

The LDAS database system involves the following software components:

- DB2 database server
- metadataAPI
- ligolwAPI
- LDAS web server
- managerAPI

The performance of the system for data insertion and retrieval will depend in part on the CPU power of the computer(s) on which the first four of these run. The disk technology used to store the database information will probably be even more important, at least after a large amount of data has been inserted. At present it is unclear what the final system configuration will be, e.g. whether the disk technology will be software RAID using conventional SCSI disks, fibre-channel RAID, etc. For now, we will take the baseline plan to be to have the DB2 server and metadataAPI running on a Sun E450, with the database contents stored on SCSI disks with software RAID (0+1), and with the ligolwAPI, LDAS web server, and managerAPI running on a second E450. This may be revised before the MDC is performed, based on further investigation of the options and/or availability of prototype or production RAID systems.

The DMT system consists of two Sun E450s at Hanford and one at Livingston, all of which receive data in real time from the data acquisition system. A set of software libraries has been

written to facilitate the creation of monitoring programs, which generally operate on the real-time data and produce either “triggers” (from transient detection algorithms) or “performance characterization” (summary information) which can be stored in the database. This MDC primarily involves the DMT library code to construct database entries and the “trigger manager” process which sends the entries to the LDAS database. Other parts of the DMT libraries may be used in the testing process, but will not themselves be rigorously tested.

3 METHODOLOGY

Testing will be automated as much as possible through the use of scripts which send commands to the database system and check the results (job success/failure, correctness of output, elapsed time, etc.). This will allow rapid re-verification of the system after bug fixes, configuration changes, etc.

Documentation should be checked for completeness and correctness by at least one person other than the author(s).

An electronic log will be used to keep track of the status of each task. All significant progress, procedures, test results, problems encountered, etc. should be recorded in this form. In addition, a few documents will be written to summarize this information, as described in the next section.

4 EXPECTED OUTCOME

Upon the successful completion of the MDC, and assuming that the system functions correctly and is stable, the database system shall be certified as operational. This implies a commitment to keep the system available as much as possible and to preserve the information recorded in the database through backups, transfer to new server hardware if ever needed, etc. The LIGO user community is likely to respond by making more active use of the database, e.g. by asking how to start inserting real information into database tables which had previously gone unused; LDAS personnel should be prepared to assist users in this effort.

The MDC can be executed “successfully” and yet reveal significant shortcomings in one or more aspects of the system. If this happens, then these issues will need to be addressed and fully tested (possibly with a follow-up MDC covering the relevant aspects) before the system can be certified as operational. On the other hand, minor shortcomings which do not seriously interfere with productive use of the database (such as borderline performance, problems which can be circumvented by internal workarounds, etc.) should not necessarily delay certification, and may be fixed incrementally. Similarly, shortcomings in any of the individual user interface tools should not hold up certification of the core database system.

It is envisioned that the results of this MDC will be reported in three documents:

1. A report on the LDAS hardware and software components.
   This should be a formal report, stating what was tested, evaluating whether the system has the required functionality, presenting performance information and evaluating whether it meets LIGO’s needs, and evaluating the robustness of the system. It should refer to other documents containing more detailed information about system design, requirements, specifications, etc. whenever appropriate. It should clearly point out any shortcomings needing to be addressed

by LDAS personnel, e.g. bugs or quirky behavior, poor performance, or additional functionality needed.

2. A summary of the user interface tools.
   This should be a brief, informal report to make LSC members aware of the available interface tools and their capabilities. It should also discuss file formats or else refer to other existing documentation. It may, in fact, be largely written in advance of the MDC, but it seems appropriate to collect and distribute this information at the time when the database system is certified as operational.
3. A Guide to LDAS Database Administration.
   This informal document should record the procedures for administration tasks relevant to LIGO, e.g. how to add a column to an existing table, how to create a tablespace on a RAID array with appropriate parameters, how to back-up and restore the database, etc. It will serve as a "how-to" guide for LDAS database administrators, and should include discussion of practical matters (things which don't seem to work as claimed, things to watch out for, etc.) where appropriate.

It will not be necessary to produce a document reporting on the validation of the database table design. Any changes found to be necessary will simply go into a revised version of the existing database table document (LIGO-T990101).

Any issues specific to the insertion of trigger data generated by the DMT system are of little interest to the general user and will be worked out internally. The DMT software documentation should contain the information needed to maintain this system in the future.

5 VALIDATION OF DATABASE TABLE DESIGN

The database tables are manually instantiated in the DB2 database by a database administrator running scripts containing SQL statements. DB2 checks the self-consistency of these table definitions, so successful instantiation ensures that the tables are well defined. Of course, their suitability for LIGO use must be judged by the scientists who wish to use them. Some tables will be in active use by the time of the MDC, but others will not, so the MDC will not attempt to certify all tables from a scientific standpoint. In addition, it is likely that new tables will be added in the future as new needs arise, e.g. tables to record information about earthquakes or gamma-ray bursts; such tables probably will not have any interaction with the existing tables, and so there should be no problem with adding them.

6 CORE DATABASE SYSTEM FUNCTIONALITY

Tests of functionality are intended to determine whether the system behaves "correctly". The correct behavior may be defined by "requirements" documents, design documents, and/or on-line documentation. Ideally these would all be consistent, but since implementations evolve, the on-line documentation is most likely to describe how the system is currently expected to behave. If there is a substantive disagreement, it should be discussed in the final report.

6.1. Communication with managerAPI

Job submission and result reporting should operate as advertised, of course. In particular, all of the methods for reporting results (parameter "–returnprotocol" in LDAS user commands) should be tested: http, ftp, mailto, file, port. In addition, the managerAPI should handle the following error conditions appropriately (normally by returning an informative error message to the user):

- Poorly formed command (e.g. missing close-brace)
- Incorrect LDAS username and/or password
- Missing a required parameter
- Invalid –returnprotocol specification
- Mal-formed user command

6.2. Data Insertion

Generally, information to be inserted into the database is first written to a file in "LIGO light-weight" (LIGO_LW) format, which is a specific XML format. An LDAS job is then run to read in the file, parse it, and do the actual insertion into the DB2 database. Besides the basic XML specification and the LIGO_LW document type definition, there are some additional LIGO-specific formatting requirements, such as the quoting of text items in the STREAM object and the treatment of special characters. In addition, there are LDAS conventions regarding the assigment of "unique ID" values in various tables. These requirements must be fully documented before the database can be certified as operational (although user interface software tools should, in most cases, save users from having to know too many of the nitty-gritty details).

The LDAS system also accepts database input in ilwd format, either over a data socket or from a file. Transmission over a data socket is actually done within LDAS (from the lwAPI to the metadataAPI) whenever a user submits a LIGO_LW file for insertion, so this does not have to be tested separately. However, reading from and ilwd file must be tested to ensure that it works and that the formatting rules and conventions are documented correctly.

Even if a LIGO_LW or ilwd file is formatted correctly, there are relationships ("referential integrity constraints") between database tables which must be obeyed by the data being inserted. For example, each entry in the `gds_trigger` table must have a `process_id` value corresponding to an entry in the `process` table. The DB2 database will reject data which does not satisfy these constraints; this situation must be handled gracefully, with an appropriate error message sent back to the user.

Functionality tests must check for correct handling of various special cases, including:

- Multiple related or unrelated tables in the same input file
- Special characters in input data
- BLOBs (Binary Large OBjects) of various sizes, up to the allowed maximum size

Error handling tests should include the following:

- Input file not found at the expected place

- Input file is not LIGO_LW
- Input file looks like LIGO_LW, but is ill-formed (e.g. missing a container level)
- Input file is incomplete, i.e. properly formatted up to a point but ends prematurely
- Input file specifies a table which does not exist in the database
- Input table includes a column which does not exist in database table
- Input table is missing a required non-null value
- Input column specified with wrong type
- Input file has special characters which are not properly escaped
- Input file is formatted correctly, but violates a DB2 referential integrity constraint
- Input file contains a BLOB which is larger than the allowed maximum size

6.3. Database Queries

The native language of DB2, as well as several other relational database systems, is SQL (for "Structured Query Language"). Most LIGO usage of the database will involve relatively simple SQL queries, and while SQL allows very complex queries to be constructed, the MDC is not designed to check DB2's handling of SQL. We will concentratate on the mechanics of submitting a query and receiving the results.

One specific thing to be tested is the handling of special characters in the SQL query. Also, the LDAS getMetaData command supports a few different protocols and a few different formats for returning the results, and all of these should be tested.

Error handling tests should include the following:

- Poorly formed SQL
- Well-formed SQL, but does not match any rows in database (shouldn't produce an error, but just return an empty table)
- Query contains an SQL statement which attempts to modify the database

7 CORE DATABASE SYSTEM PERFORMANCE

The most obvious performance metric is the time taken to complete a database transaction. This will depend on the data volume, the nature of the data (e.g. the data types involved), the amount of data stored in the database, whether the input/output is in LIGO_LW or ilwd format (since LIGO_LW involves an extra translation step within LDAS), and the load on the LDAS system. The MDC should explore various combinations of these conditions. In particular, it should characterize the effect of multiple concurrent jobs, since data insertion and queries need to be able to go on simultaneously.

A second concern is whether there are practical limits on the volume of input or output data, e.g. from the finite memory sizes of the computers on which the LDAS APIs run, or from a limit on the number of data rows which can be handled. The MDC should check for any such limitations.

Finally, long-term stability is an essential requirement for the database system. Ideally, the system should run continuously without human intervention even when heavily used. This may involve automatic exception handling, e.g. re-starting APIs which have grown to use up too much memory, but it is critical that no data be lost in the process. In any case, a notification system should be in place to warn LDAS personnel about any failures or significant error conditions.

Some specific issues are discussed in the sections below.

7.1. Data Insertion

There has been no formal statement of the required sustainable data insertion rate, but one entry per second seems like a reasonable goal. (If the average size of an entry is 1 kb, this works out to about 30 Gb per year; the average entry size might be larger, if we store a lot of spectra or other large objects.) Of course, there will be queries going on at the same time, so it is desirable to have a large safety margin.

The metadataAPI normally assigns unique-ID values to input records which need them. This may be rather time consuming, so timing tests should be done with and without this step.

If there is a limit on the size of the input file which can be handled, it is probably better for the system to gracefully refuse to process it and return an informative error message, rather than crashing.

7.2. Database Queries

The time required for the system to execute a database query will vary widely depending on the complexity of the query and on how well the indexes maintained by DB2 may be used to optimize the processing. Therefore, “realistic” queries should be used to evaluate the speed which LIGO users may expect.

The handling of large output tables is a particular concern, since users do not necessarily know how much data will be returned from a given query. LDAS provides a mechanism to limit the number of rows which can be returned, which is fine, but there may also need to be a check on the total data volume in bytes, particularly for tables with BLOBs.

8 DMT DATA INSERTION PROCEDURE

The DMT system has a “trigger manager” to facilitate the insertion of data into the database. A DMT monitor program uses C++ calls initially to define the structure of a table and later to add rows. The information for each row is transmitted to the Trigger Manager process, which serves as a gateway and a buffer; it periodically creates a LIGO_LW file from the information it has received, and submits it to LDAS to be ingested into the database.

The DMT software must permit rows to be created for several different tables: `process`, `process_params`, `filter`, `filter_params`, `gds_trigger`, `sngl_datasource`,

`sngl_transdata`, `summ_value`, `summ_statistics`, `summ_spectrum`, `summ_comment`, `segment_definer`, and `segment`. If the LDAS database is temporarily off-line, the trigger manager must be able to accumulate files and then automatically insert them when the database becomes available again.

9 USER INTERFACES

While not part of the database system itself, user interfaces are needed to make it useful. This section describes the user interface tools which should be in place by the time of the MDC; other tools may come later.

The following should exist in a stable form and be documented:

- `putMeta`, a command-line utility to submit an arbitrary LIGO_LW file to the LDAS database.
- `getMeta`, a command-line utility to submit a database query and retrieve the results to standard output or to a file.
- `guild`, a graphical user interface to construct and submit a database query and to display the results. (Also can save the results as LIGO_LW or in various ASCII formats.)
- `Xlook`, a graphical user interface to display the contents of any LIGO_LW file (including data objects other than tables).
- A Matlab interface to submit a database query and retrieve the results into Matlab arrays. There should also be some scripts/functions to help manipulate the table data in this form, e.g. text comparisons with wildcards (and possibly case-insensitive). A function to correlate two tables based on time is also essential.
- A LAL-compatible C package to submit a database query and to read/write table data sequentially. (May use an existing XML library.)
- A C++ library to submit a database query and to read/write table data sequentially. This is likely to just use the C package, possibly with a wrapping layer to make it more convenient.

All of these interfaces, except possibly `guild` and `putMeta`, will not communicate with LDAS directly but will go through a “data flow manager” (dfm) process running on the local machine. The dfm will take care of comminicating with the LDAS managerAPI and retrieving the results, in order to provide a simplified interface for all other programs.

10 DATABASE ADMINISTRATION TASKS

The following tasks should be exercised before and/or during the MDC, and the procedures should be detailed in the “Guide to LDAS Database Administration” which is one product of the MDC. Other procedures may be included too if deemed appropriate.

- Acquire and apply DB2 fix pack.
- Create tablespace on RAID (?) disk system, with optimal parameters.

- Add a new table.
- Add a column to an existing table.
- Create a new index on an existing table.
- Remove “erroneous” database entries associated with a given process.
- Perform backup (procedure, time taken)
- Perform restore (procedure, time taken)
- Add a new disk system to an existing database

Furthermore, beginning with the MDC (or sooner), each database installation should be backed up on a regular basis, preferably with an automated procedure.

11 TASK TABLES

In general, the person who produces a software component or performs a certain test is responsible for writing the corresponding documentation.

Table 1: MDC Organization and Infrastructure Tasks

Task	Who	Time estimate	Comments
Finalize target date for MDC	Peter	1 day	Need to coordinate with CDS and DMT availability, and avoid engineering run
Create test-script parser & checker	Peter?	5 days	Uses putMeta, getMeta

Table 2: Database Table Design Validation Tasks

Task	Who	Time estimate	Comments
Finish revising DB table design and updating document	Peter	3 days	Need feedback from Julien, others?
Instantiate revised table designs, check for success		1 day	
Populate revised tables with fake data		2 days	
Update DB table design document, if necessary	Peter	1 day	
Check DB table design document for completeness, usefulness		2 days	

Table 3: Core Functionality Tasks

Task	Who	Time estimate	Comments
Test all return protocols, check documentation		1 day	
Test managerAPI handling of various error conditions		1 day	
Write documentation about LIGO_LW formatting and conventions for DB insertion	Masha / Mary?	2 days	
Check documentation about LIGO_LW formatting for accuracy, completeness		2 days	
Write documentation about ilwd formatting and conventions for DB insertion	Masha / Mary?	2 days	
Check documentation about ilwd formatting		2 days	
Test insertion using various input files (some with multiple related or unrelated tables, symbolic vs. explicit unique IDs, LIGO_LW vs. ilwd, etc.)		7 days	
Test handling of special characters in input data		1 day	
Test insertion of BLOBs of various sizes		1 day	
Check handling of various error conditions when inserting data		2 days	
Check handling of various errors in SQL queries		1 day	
Evaluate functionality, identify any shortcomings		1 day	

Table 4: Performance Tasks

Task	Who	Time estimate	Comments
Measure insertion time for various tables, various numbers of rows, LIGO_LW vs. ilwd, symbolic vs. explicit unique IDs; graph and summarize		5 days	
Measure query time for various tables, various numbers of rows, LIGO_LW vs. ilwd; graph and summarize		2 days	
Evaluate effect of concurrent insertion & querying		2 days	
Populate database with a large amount of fake data		2 days	
Check for limits on input file (number of rows, size in bytes)		2 days	Exceeding limit should be handled gracefully
Check for limits on output file (number of rows, size in bytes)		2 days	
Run long-term stability test		5 days	
Cause hard failures (out of memory, sudden reboot, power loss); check for data loss, notification, restarting procedures		2 days	
Evaluate performance		1 day	Did MDC use target CPUs and disk technology?

Table 5: DMT Data Insertion Tasks

Task	Who	Time estimate	Comments
Add ability to create entries for all relevant tables	John	4 days	
Replace ldas_submit script with a smart program (probably the Data Flow Manager) to buffer data if LDAS is down	Peter	3 days	
Write dummy monitor to generate pseudo-random triggers at a controlled rate		1 day	
Test throughput of Trigger Manager		1 day	
Create example trigger monitor		2 days	

Table 5: DMT Data Insertion Tasks

Task	Who	Time estimate	Comments
Create example performance monitor		2 days	
Set up a DMT computer to be dedicated to the MDC	John	1 day	
Set up several monitors to run continuously during long-term test		5 days	
Ensure that user documentation for trigger/table generation API is up-to-date, including example code		2 days	

Table 6: User Interface Tasks

Task	Who	Time estimate	Comments
Create Data Flow Manager; write documentation	Peter	6 days	
Create putMeta and getMeta utilities; write documentation		2 days	Use Data Flow Manager
Write documentation about guild		1 day	
Update Xlook for release?; write or update documentation?		3 days?	
Create Matlab interface to DFM		5 days	Probably use C package for table i/o
Create Matlab tools to select table rows (e.g. text comparison)		5 days	
Create Matlab function to correlate events in two different tables		3 days	
Create Matlab script to display spectra from database		3 days	
Write documentataion about Matlab interface and tools, with examples		4 days	
Create LAL-compatible C package for table i/o		6 days	May use an existing XML library
Write documentation and sample programs for C interface		4 days	

Table 6: User Interface Tasks

Task	Who	Time estimate	Comments
Create C++ i/o library, or wrap the C library		6 days	
Write documentation and sample programs for C++ interface		4 days	
Assemble summary of user interface tools (with contributions from tool authors)		3 days	

Table 7: Database Administration Tasks

Task	Who	Time estimate	Comments
Document installation/upgrade procedure		1 day	Ed and Peter have notes already
Acquire & apply DB2 fix pack		1 day	
Exercise & document procedures for database setup (DB creation, tablespace creation, table creation, etc.)		3 days	Tablespace parameters should be tweaked if using a RAID system
Exercise & document procedures for database configuration changes (add column to existing table, create a new index, etc.)		2 days	
Establish script/procedure for deleting entries associated with a given process		2 days	
Add a new disk system to an existing database		1 day	
Establish DB backup procedure, and start doing it regularly		4 days	Probably should be automated
Exercise & document restore operation		1 day	
Assemble Guide to DB Admin		5 days	

Table 8: Summary and Evaluation Tasks

Task	Who	Time estimate	Comments
Assemble formal report on tests of LDAS hardware and software components		8 days	Should clearly point out shortcomings, additional functionality needed, etc.
Evaluate completeness and success of MDC			
Inform LSC of results of MDC			
Arrange follow-up MDCs, if necessary			

12 ISSUES NOT ADDRESSED BY THIS MDC

Replication of data between sites (should this be added back in???)

Procedure for inserting astrophysics event candidates (will be part of MPI template search MDC)

Use of database as catalog of raw data in archive (part of archive MDC, involving metadata generated during ingestion process) -- i.e. frameset-related tables.