LSC Segment Database

Duncan Brown
Caltech

LIGO-G060176-00-Z
Goals of Segment Database

- Make interferometer state information available rapidly for S5 searches
- Allow automated insertion of DQ flags from DMT
- Reduce latency of releasing human generated data quality information
- Replicate DQ information between Caltech and Observatories
- Provide simple interface for accessing data that integrates with existing tools
Architecture

• **Use IBM DB2 database for underlying engine**
  » Reliable, well tested, runs on Solaris and Linux

• **Write LSC specific client/servers in Python**
  » Good interface between Python and DB2 (mXODBC)

• **Base table design on existing database tables**
  » Table design modified for using in segment database

• **Use IBM Q-Replication for replication**
  » Straightforward to set up a 3 element peer-to-peer network
Database Implementation

- **Table designed based on existing table design**
  - All inserts must have a process table
  - The `segment_definer` table describes a segment
  - The `segment_def_map` tables maps segment definitions to intervals
  - The `segment` table contains the start and end times

- **Also have an LFN table and a grid_cert table**
  - Track which frame file state information came from using LFNs
  - Track who inserted data with grid_cert table

- **Otherwise straightforward DB2 database**
  - DB2 instance created under the user lbd on gateway machines
Replication

• Use IBM Q-Replication to set up peer-to-peer replication

• WebSphere MQ messages queues set up between CIT, LHO and LLO. At each site:
  » Two transmit queues (e.g. LHO_TO_LLO, LHO_TO_CIT)
  » Two receive queues (e.g. LLO_TO_LHO, CIT_TO_LHO)
  » Plus a couple of control queues

• Each site runs a capture server and an apply server
  » Capture sever pushes all incoming transactions into xmit q
  » Apply server gets transactions from recv q and applies to database

• Capture and apply servers store state in ctrl tables in database
  » Set up when database was created
Direct Clients

- Direct clients run on gateway machines and connect directly to the database.
  - Can be run as user ldbname or grid

- segpagegen
  - Creates nightly web pages containing segment dump
  - Segment dump written to ascii web pages for other tools to grab
  - Client program in glue/sbin/segpagegen installed on gateway
  - Run from ldbname user’s crontab at 1600 UTC daily
    - Runs from shell script that sets up environment

- statedb.py
  - Python module to publish IFO state information into database
  - Used by Ben’s publishing scripts
  - Also used by publishstatefromfile and bulkpublishstate scripts
Grid Servers

• Implemented as python modules and run by Idbdd program
  » Idbdd provides common infrastructure (logging, gsi socket server, etc.)
  » Loads one of the two module below to handle connections

• LSCsegFindServer.py
  » Receives segment query on socket from client
  » Constructs SQL and executes it to get results from database
  » Returns output to client for display/writing to file
  » distinct() method
    – Returns available segments (with meanings)
  » segmentFindWithMetadata_vx()
    – Accepts query based on interferometer, type, start time, end time
    – Multiple types are unioned
    – Multiple IFOs are intersected
Grid Servers (continued)

• **LDBDServer.py**
  » Provides map between LIGO_LW XML and database
  » Reads configuration from `/export/ldbd/etc/ldbdserver.ini`
  » Reads database table design from DB2 on startup
  » `query()`
    – Executes SQL query on database and returns results as XML
  » `insert()`
    – Takes LIGO_LW XML that complies with table design and inserts it
  » `insertmap()`
    – Same as `insert()` but adds an LFN -> PFN map to an RLS server
  » `Insertdmt()`
    – Same as `insert()` but handles XML files containing existing `segment_definer` rows and updates process table with current end time of DMT process, if it already exists in the database
  » All insert methods capture the users DN and insert it in the `grid_cert` table
Grid Clients

- **LSCsegFind.py**
  - Straightforward interface to LSCsegFindServer
  - API directly by onasysd
    Command line version is LSCsegFind

- **LDBDClient.py**
  - Straightforward LDBDServer client for query(), insert() and insertmap()
  - API used directly by dmtdq_seg_insert and LSCdqInsert
    Command line version is ldbdc

- **dmtdq_seg_insert**
  - Used by DMT to insert XML containing online DQ
  - Calls LDBDClient.py with insertdmt() method
  - Deletes XML files on successful insert

- **LSCdqInsert**
  - Takes a list of GPS start/stop times, constructs XML and sends it to LDBDServer
    for insert into database
More on Table Design

- All tables have a creator_db column for replication
  - LHO 1, LLO 2, CIT 3

- segment_definer is constrained to have unique rows for (run, ifos, name, version)
  - e.g. S5,H1,DUST,1

- segment_def_map table links definitions to segments
  - (creator_db, segment_def_id) mapped to (creator_db, segment_db)

- segment table contains [gpsstart, gpsend]
  - Each segment has an active column: 0 for off, 1 for on
  - Science segments also populate segnum column with the science segment number
Problems encountered during S5

• Replication shuts off if:
  » Large clock skew between two sites
    – Fix clock skew and restart replication
  » Conflict between tables
    – Rare, but could happen if user inserts DQ segments at LHO and the immediately inserts the same data at LLO before original insert can be replicated from LHO to LLO
    – Solution is to delete conflicting insert by deleting process table and restart replication
  » Replication message goes missing
    – Very rare: only happens if bad crash on server (e.g. raid crash at CIT)
    – Fix is to re-sync tables and restart replication

• Python servers have died a couple of times during S5
  » Just restart, diagnosing error in /export/ldbd/var/log if possible
To Do List

- Extend LSCsegFind.py API to query active flag for segments
  » Would allow users to take intersections with “DQ flag not active”
- On site backups / deal with DB log files.
  » Currently the databases are not backed up. We rely on replication being working to ensure that a copy of the contents of the database are available, if there is a catastrophic failure at a site. Database should really be backed up locally as well. Cancerous log files should be dealt with.
- Status lights for the servers at CIT.
  » The status of the LSCsegFind LDBD and Trigger servers at CIT should be available on the status pages.
- Status for segment web dumps.
  » It would be nice to have status light which indicates if the latest segment web dump is older than 27 hours, which would indicate whether the cron job that creates these pages is functioning or not.
- Better status information for replication.
  » A red/green light for the Q replication processes would be very useful.