Realtime Event Notification

FlexCDC and FlexCDC plugins

Justin Swanhart

PLMCE 2015

Introduction

- Who am I?
- What do I do?
- What is this talk about?

Flexviews and FlexCDC

- Flexviews – Materialized view toolkit
- FlexCDC – Change data capture
  - Reads binary logs
  - Writes changes into change log tables
  - Can use plugins to send data to other systems

FlexCDC

- CDC – Change Data Capture
  - Uses mysqlbinlog to read from remote server
  - Decodes ROW based binary logs
  - Uses change log tables to interpret RBR (data types / column names)

FlexCDC requirements

- server_id must be set in my.ini
- binlog_format=ROW (not MIXED or STATEMENT)
- Row images must be FULL
- Must have user with following privs
  - REPLICATION SLAVE
  - REPLICATION CLIENT

FlexCDC requirements continued

- PHP 5.3+
- Pear (for getopt)
- PHP-MySQL

FlexCDC setup

- Clone github.com/greenlion/swanhart-tools
- FlexCDC in flexviews/consumer
- Create ini file
  - consumer.ini.example has necessary settings
- Setup FlexCDC
  - setup_flexcdc.php

consumer.ini (subset)


```
[flexcdc]
mysqlbinlog=/usr/local/mysql/bin/mysqlbinlog
database=flexviews
binlog_consumer_status=binlog_consumer_status
mvlogs=mvlogs
mview_uow=mview_uow
[source]
user=root
host=127.0.0.1
port=3306
password=
[dest]
user=root
host=127.0.0.1
port=3306
password=
```

Run consumer

- Includes “angel” script like mysql
- Run consumer_safe.sh &
- Runs in background and captures changes

Using standard changelog tables

- Two ways to add changelogs to tables
  - Use add_table.php script to add tables to log
  - Install flexviews setup.php and use flexviews.create_mvlog()

Capturing changes

- FlexCDC uses mysqlbinlog
- Decodes RBR to pseudo-SBR
  - ### INSERT INTO `test`.`demo`
  - ### SET
  - ### @1=1
  - ### @2=10
  - # at 124794

Capturing changes

- mysql> call flexviews.create_mvlog('test','demo');
- Query OK, 1 row affected (0.01 sec)

Determining log table

- mysql> select * from mvlogs where table_name='demo'\G
- *************************** 1. row ***************************
- table_schema: test
- table_name: demo
- mvlog_name: mvlog_d04c8f0c8097ee8bce23318fb44950dc
- active_flag: 1
- 1 row in set (0.00 sec)

Log table


```
mysql> select * from mvlog_d04c8f0c8097ee8bce23318fb44950dc;
+----------+--------+--------------+--------+------+------+
| dml_type | uow_id | fv$server_id | fv$gsn | c1   | c2   |
+----------+--------+--------------+--------+------+------+
|        1 |      7 |           99 |      2 |    2 |   20 |
+----------+--------+--------------+--------+------+------+
1 row in set (0.00 sec)
```

Columns added by FlexCDC

UOW (transaction information) table


```
mysql> select * from mview_uow;
+--------+---------------------+---------+
| uow_id | commit_time         | gsn_hwm |
+--------+---------------------+---------+
|      1 | NULL                |       1 |
|      2 | 2015-04-15 06:08:40 |       1 |
|      3 | 2015-04-15 06:08:40 |       1 |
|      4 | 2015-04-15 06:08:40 |       1 |
|      5 | 2015-04-15 06:09:53 |       1 |
|      6 | 2015-04-15 06:11:11 |       1 |
|      7 | 2015-04-15 06:11:52 |       2 |
+--------+---------------------+---------+
7 rows in set (0.00 sec)
```

UOW_ID and GSN

- Each transaction is assigned a UOW_ID
- Each row change a unique GSN (global sequence number)

Transactional changes group together


```
mysql> delete from test.demo;
Query OK, 2 rows affected (0.00 sec)

mysql> select * from mvlog_d04c8f0c8097ee8bce23318fb44950dc;
+----------+--------+--------------+--------+------+------+
| dml_type | uow_id | fv$server_id | fv$gsn | c1   | c2   |
+----------+--------+--------------+--------+------+------+
|        1 |      7 |           99 |      2 |    2 |   20 |
|       -1 |      8 |           99 |      3 |    1 |   10 |
|       -1 |      8 |           99 |      4 |    2 |   20 |
+----------+--------+--------------+--------+------+------+
3 rows in set (0.00 sec)
```

Cleaning up changelogs

- Change logs are analogous to binary logs, except the changes are stored in a table
- Like binary logs, it is necessary to purge data from the change logs or they will grow unbounded

Log_retention_interval

- Setting in consumer.ini which controls for how long change log entries are retained
- Use INTERVAL syntax, ie “10 day” (the default)
- Run cleanup_history_safe.sh
- Every 600 seconds wakes up and cleans up old rows

ALTER TABLE?

- When an ALTER is processed from binary log if the table is being logged, the ALTER is applied to the log table (if applicable)
  - Indexes are not applied
  - ROW_FORMAT, KEY_BLOCK_SIZE, etc, not applied
  - Only column names and data types basically

Using FlexCDC with Flexviews

- FlexCDC is part of Flexviews
- Flexviews uses change log tables to efficiently update materialized views
- You can’t change FlexCDC table names if you want to use Flexviews

Using the changelog tables with ETL

- ETL (extract transform load) tools are typically designed to work with change data capture
- Examples include Pentaho, Jaspersoft, almost any BI tool

How popular online clothing site uses FlexCDC

- Use FlexCDC “auto_changelog” mode
  - Automatically creates log tables for any table that has data changed by the database
  - FlexCDC captures changes to all tables

Using with ETL tool

- ETL tool remembers the highest UOW_ID that it has read
- Gets lists of tables to collect from from *mvlogs* table
- SELECT from the log tables where UOW > last_uow_id

ETL

- ETL tool is responsible for making changes in downstream database (Redshift for example)
- ETL tool can also maintain aggregate tables, etc, in downstream database
- One way only – changes in downstream do not propagate upstream

Using ETL flows

- If you capture changes from subset of tables you can use normal ETL flow
- Read changes (as inserts and deletes) from log table, applying changes to downstream table (data warehouse)
- Transform along the way as required

Gearman and gearman UDF

- Gearman is a job server/message queue
  - Workers – Wait for job input, process job, return results
  - Servers – Accept jobs from clients, send to workers. Marshalls data between worker and client
  - Clients – sends input to workers, gets results

Setup steps

- Install gearman server
- Create a worker script to do the work you need (insert into remote database for example)
- Create a client

UDF client

- Could be FlexCDC plugin (more on this shortly)
- But can be Gearman MySQL UDF as well
  - Compile with libgearman
  - Call gman_servers_set(…) to connect MySQL to Gearman (good to put in init_sql command)

Using the Gearman UDF


```
set v_args := concat('{"sql":"', v_sql,
'","schema_name":"',v_remote_schema,'"}');

set v_json := gman_do('shard_query_worker', v_args);

SELECT gman_do_background('worker_name', 'arguments');
```

Use FlexCDC + Gearman UDF

- Create changelog as normal
  - Use add_table.php for Flexviews.create_mvlog()
- Add TRIGGER to log table
  - AFTER INSERT trigger probably is best
  - Use gman_do_background to send changes to workers

Pros/Cons

- Pros
  - Easy to set up
  - Worker code can be any language
- Cons
  - Triggers
  - Requires a trigger per log table, not generally useful for many tables
  - Each row change is sent to Gearman (no batching) not really transactional

Using FlexCDC plugins

- FlexCDC supports plugins since early last year
- Plugins are written as a PHP class which is loaded by FlexCDC at startup
- Multiple plugins are supported
- You can declare plugin execution order

Still need source and dest

- When using plugins, FlexCDC maintains an empty copy of the source table
  - Allows FlexCDC to provide proper data types
  - Allows FlexCDC to provide column names

Enabling plugins in the consumer.ini


```
;The example plugin prints out all the rows that are sent
to it - THEY ARE NOT LOGGED INTO THE MVLOG TABLE
;plugin=example_plugin.php

;You can load more than one file with plugins:
plugin=file1.php,file2.php,file3.php

;Of course, if you want more than one plugin, each MUST
have a unique name (again this is the CLASS name in the
file(s) loaded above)
plugin_order=FlexCDC_plugin,Class1,Class2
```

Plugins implement FlexCDC_plugin_interface


```
interface FlexCDC_Plugin_Interface {
        static function plugin_init($instance);
        static function plugin_deinit($instance);
        static function begin_trx($uow_id, $gsn, $instance);
        static function commit_trx($uow_id, $gsn, $instance);
        static function rollback_trx($uow_id, $instance);
        static function insert($row, $db, $table, $uow_id, $gsn, $instance);
        static function delete($row, $db, $table, $uow_id, $gsn, $instance);
        static function update_before($row, $db, $table, $uow_id, $gsn,
$instance);
        static function update_after($row, $db, $table, $uow_id, $gsn,
$instance);
}
```

Init/Deinit


```
static function plugin_init($instance);
static function plugin_deinit($instance);
```

These functions are called during plugin startup and shutdown. If you are sending changes to an external data store, this is a good place to connect/disconnect from the remote server

Instance is a copy of the FlexCDC object (not generally useful here, but present for completeness)

Transaction state management


```
static function begin_trx($uow_id, $gsn, $instance);
static function commit_trx($uow_id, $gsn, $instance);
static function rollback_trx($uow_id, $instance);
```

These functions manage transaction state.

$uow_id – The transation identifier assigned by FlexCDC for the transactions

$gsn – GSN high water mark (highest GSN used in transactions)

$instance – FlexCDC instance for helper functions

Row changes


```
static function insert($row, $db, $table, $uow_id, $gsn,
$instance);
static function delete($row, $db, $table, $uow_id, $gsn,
$instance);
static function update_before($row, $db, $table, $uow_id,
$gsn, $instance);
static function update_after($row, $db, $table, $uow_id,
$gsn, $instance);
```

These functions manage transaction state.

$row – the row data (associative array includes column names)

$db – the schema that the change originated in

$db – the table that was changed

Sending changes to non-transactional stores

- FlexCDC provides $uow_id and $gsn
- Record $gsn in remote system
- Write plugin such that if “something bad” happens, it can skip through highest GSN
- GSN are monotonically increasing, never skip values, and are never reused

Sending changes to transactional stores

- Start a transaction against remote store in begin_trx(..)
- Commit in commit_trx(…)
- Rollback in rollback_trx
- (pretty simple)

Batching changes

- Instead of writing to remote store for every row change use a class member to hold changes and then batch them into the remote database in commit_trx(…)
- Similar technique can be used to send UPDATE statements instead of delete/insert in before_update and after_update

Advantages over other solutions

- MySQL C binlog API, Ruby binlog API, Tungsten, etc
  - Can not provide column names
  - Can not correctly interpret unsigned data types
  - Do not use binlog_dump command to ‘act as a slave’
  - Do not handle ALTER TABLE

Possible use cases

- Keep Sphinx/SOLR up to date based on data changes
- Invalidate Memcached based on row changes
- Turn row changes into SELECT statements to warm a slow slave
- Send data to Kafka, RedShift, Hadoop, any third party data store with PHP client

Q/A

http://ceilingcat.ninja