A Step Toward a True Server-Client Datascope Capability: EVServer and EVClient

Danny Harvey
Boulder Real Time Technologies, Inc.
Antelope User Group Meeting, ZAMG, Vienna
2017 May

Outline

- Description of problem
- A strategy for a Datascope implemented server-client database software utility
- Antelope 5.7 **`EVServer`** and **`EVClient`** classes plus python extensions

Legacy Datascope

- Originally designed to provide high performance database management operations on static databases (primarily as a research tool).
- An embedded approach to manipulating database tables and views.
- Support for multiple client interactions with database largely accomplished through ad hoc coordination
- Ubiquitous use of virtual memory mapping

BRTT

Kinemetrics

Datascope pros

- Very high performance with a small footprint, equal to or better than Oracle, MySql, Ingress, Postgress, etc.
- Provides complete relational database management functionality with many interfaces and without the use of Sql.
- Does not require a server. Very easy for users to quickly interact with databases in many different ways.
- Still today one of the best database systems for researchers.

BRTT

Kinemetrics

Datascope cons

- Does not provide a Sql interface.
- In theory, requires all database interaction be local to the database files.
- Does not provide comprehensive, safe and automated database synchronization between multiple client processes.
- If not managed properly, can cause problems in a network operations environment.

BRTT

Kinemetrics

Operational Environment

- **Automated processing that is continuously updating many database tables often.**
- **Simultaneous interactive manual review by analysts that also causes updating of many database tables.**
- **Simultaneous administrative tasks, such as `rtdbclean`, either automated or manually run, that can significantly modify database tables.**

BRTT

Kinemetrics

- **Datascope provides basic access to database fields and records through pointers into virtual memory mapped files.**
- **When one process modifies a database file, it is very easy for other processes to have invalid references to the database fields.**
- **The embedded nature of Datascope does not allow for inter-process, or even inter-thread synchronization.**

Practical Considerations

- Not all database tables change frequently.
- The meta-database tables, `site`, `sitechan`, `sensor`, `instrument`, `calibration`, etc., change infrequently. Typically these changes are accommodated by shutting the real-time system down, ceasing all analyst review operations, change the meta-database and start everything up again.
- The `wfdisc` table changes on a more frequent basis. However the `wfdisc` table is usually only modified by `orb2db` or `orb2wf` and not by any analyst review processing.
- The database tables associated with event processing change frequently, both by the automated processing and by analyst review. This processing involves the `detection`, `event`, `origin`, `origerr`, `assoc`, `arrival`, `wfmeas`, `netmag`, `stamag` and `mt` tables.

"Event View" specialized server-client interface into Datascope databases

- **Two new object oriented c++ classes have been introduced into Antelope 5.7 – `EVServer` and `EVClient` (see man EV(3)).**
- **These event view classes provide a server-client implementation of database access operations specific to the various seismic event tables in the `css` schema.**

BRTT

Kinemetrics

NAME

EV - BRTT utility for earthquake event view formation

SYNOPSIS

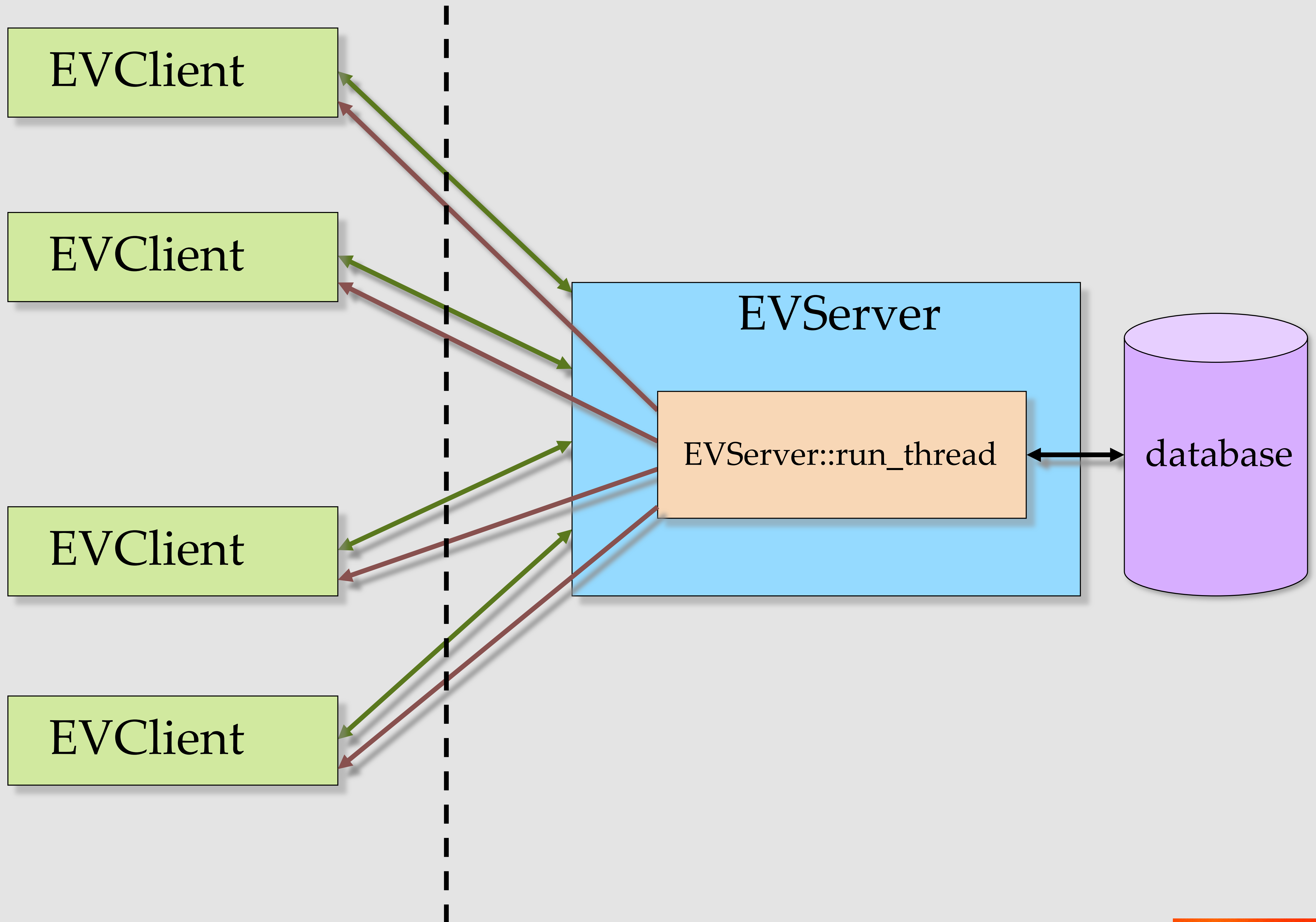
-lbrttutil

#include "EV.h"

DESCRIPTION

There are two fundamental classes, **EVServer** and **EVClient**, that implement complete views of earthquake event information from underlying databases. They are intended to be dynamic in response to changing databases. Information from events, origins, origin errors, associations, arrivals, detections, stations, magnitudes and moment tensors are joined in a set of views that can be returned through a set of specialized structures.

The underlying database is monitored and the views are made by a single **EVServer** object. The views are refreshed automatically by **EVServer** objects whenever any of the database file modification times have changed. **EVServer** makes all of the joins though calls to **dbmatches(3)** only, without using the various other Datascope view generation routines, such as **dbjoin(3)**. Most Datascope view generation routines cannot track dynamic changes in the underlying database. By only using **dbmatches(3)**, which is designed to track certain changes in the underlying database, **EVServer** objects can track changes in the database and recompute the various view structures as required. All calls to **dbmatches(3)**, **dbget(3)** or **dbgetv(3)** made by **EVServer** objects trap error returns, which could be caused by changes in the database during **EVServer** processing. When **dbget(3)**, **dbgetv(3)** or **dbmatches(3)** return errors, the **EVServer** object will automatically close the database, reopen it, free all **dbmatches(3)** hooks, and reform the various views. This will also happen automatically whenever the database files shrink in size.



BRTT

Kinemetrics

- `EVServer` objects launch a thread, `EVServer::run_thread`, to interact with the database. This thread is the only thread that interacts with the database.
- The primary responsibility of `EVServer::run_thread` is to keep an up to date internal set of structures that contain all of the information from the database, including copies of the database records, all linked together to form earthquake event oriented views.
- None of the internal structures contain database pointers or other references back to the database. In this way the internal structures are complete and self consistent snapshots of the database at the time when the structures were made.

- `EVClient` objects can request copies of the internal structures that `EVServer` objects maintain.
- `EVClient` objects can register callback functions with their `EVServer`. `EVServer::run_thread` will execute these callbacks whenever any of the internal structures have changed.
- All `EVClient` acquired event view structures are complete and self consistent snapshots of the database at the time when the structures were made.
- `EVClient` objects never try to reference the database directly.

ev_test.py — ev

ev_test.py

```python
#!/usr/bin/env python

import os
import sys
import time
sys.path.append(os.environ['ANTELOPE'] + "/data/python")

import signal

signal.signal(signal.SIGINT, signal.SIG_DFL)

from antelope.ev import *
from antelope.pfsubs import *

argc = len(sys.argv)
if argc != 1 and argc != 2 and argc != 3:
    print "usage:ev_test [dbname [refresh_interval]]"
    sys.exit (1)

if argc == 1:
    dbname = "/opt/antelope/data/db/demo/demo"
else:
    dbname = sys.argv[1];
interval = 0.0;
if argc == 3:
    interval = float(sys.argv[2])

if interval > 0.0:
    server = EVServer ( dbname, 1 )
else:
    server = EVServer ( dbname, 0 )

client = EVClient ( server )

print "Initial sizes"
print "server_events_number = %d" % client.server_events_number()
print "server_events_size = %d" % client.server_events_size()
print "server_detections_size = %d" % client.server_detections_size()
print "server_expanded_events_size = %d" % client.server_expanded_events_size()

if interval > 0.0:
    client.registercallback (lambda pfstring: mycallback (pfstring, client))
    server.auto_refresh ( interval )

    import time
    while 1:
        time.sleep (1)

```

Line 15, Column 1 | Tab Size: 4 | Python

BRTT

Kinematrics

ev_test.py — ev

ev_test.py

```python
14
15  def mycallback (pfstring, client):
16      argsd = PfSubs.pfstring2py (pfstring)
17      print
18      print pfstring,
19      print "mycallback",
20      if argsd['database_reset'] != '0':
21          print "database_reset",
22      if argsd['events_changed'] != '0':
23          print "events_changed",
24      if argsd['detections_changed'] != '0':
25          print "detections_changed",
26      if argsd['last_modified_event_changed'] != '0':
27          print "last_modified_event_changed",
28      print
29
30      if argsd['events_changed'] == '0' \
31                  and argsd['detections_changed'] == '0' \
32                  and argsd['last_modified_event_changed'] == '0':
33          return
34
35      client.grab_lock ()
36
37      if argsd['events_changed'] != '0':
38          print "server_events_size = %d" % client.server_events_size()
39      if argsd['detections_changed'] != '0':
40          print "server_detections_size = %d" % client.server_detections_size()
41      if argsd['last_modified_event_changed'] != '0':
42          print "server_expanded_events_size = %d" % client.server_expanded_events_size()
43
44      client.release_lock ()
45
```

Line 53, Column 1 | Tab Size: 4 | Python

BRTT

Kinemetrics

peanuts: /opt/antelope/dev/src/bin/python/scripted/ev — -tcsh ▸ dbpick — 64×33

```
peanuts% ev_test.py ~/rtdemo_anza/db/anza 1
Initial sizes
server_events_number = 181
server_events_size = 1443952
server_detections_size = 157376
server_expanded_events_size = 130320

instance 0
database_reset 0
events_changed 0
detections_changed 1
last_modified_event_changed 0
last_modified_event_evid 348
expanded_events_changed &Tbl{
}
mycallback detections_changed
server_detections_size = 157840

instance 0
database_reset 0
events_changed 0
detections_changed 1
last_modified_event_changed 0
last_modified_event_evid 348
expanded_events_changed &Tbl{
}
mycallback detections_changed
server_detections_size = 158304
^C
[2]  - Done                          subl ev_test.py

peanuts%
```

BRTT

Kinemetrics

Future Development

- Separate `EVServer` and `EVClient` objects.
- Provide database writing functionality.