Table of Contents

COOL vector payload.....1

Vector data in COOL: channels vs. vector payload vs. BLOBs.....3

COOL vector payload

COOL now supports different modes for storing payloads in folders:

- inline payload (the default mode)
- separate table mode
- vector payload mode

If the separate table mode is used, the payload of the IOVs is internally stored in a separate table. If this mode is used, multi version folders with large payloads will use less space in the database, since the so called system-objects don't duplicate the payload but refer to the payload in the separate table. There is no difference in the client API using this mode. This mode is succeeded by the vector payload mode, which has more features, so it probably makes more sense to use the vector payload mode, if you have the choice.

The vector payload mode is related to the separate payload mode, but in addition it allows several payload records, or even zero payload records to be associated with one IOV. The number of payload records is varialbe from IOV to IOV. The payload is also stored in a separate table, which means that this mode also has the advantage of less overhead for system-objects, but internally the separate payload mode is different from this mode.

To access the payload records in vector payload mode, the new payloadIterator() method of the IObject interface should be used, which returns an iterator similar to the record iterator. It is still possible to use the old interface using the payload() method of the IObject interface, but you will only get access to the first payload record using this method. It is not recommended to use the old interface on vector payload folders.

To store records vectors of IRecordPtr are used with a new storeObject() method in the IFolder interface.

Here is a short PyCool example for the vector folder mode:

```
#!/bin/env python
from PyCool import cool

def dataPtr(i,j,c):
  data = cool.PyCool.Helpers.IRecordPtr(  spec  )
  data.get()['i'] = i
  data.get()['j'] = j
  data.get()['c'] = c
  return data

connectString = 'sqlite://;schema=COOLTEST.db;dbname=COOLTEST'

dbSvc = cool.DatabaseSvcFactory.databaseService()
print 'dropping database', connectString
dbSvc.dropDatabase( connectString )

print 'creating database'
db = dbSvc.createDatabase( connectString )

print 'setting up spec'
spec = cool.RecordSpecification()
spec.extend( 'i', cool.StorageType.Int32 )
spec.extend( 'j', cool.StorageType.Int32 )
spec.extend( 'c', cool.StorageType.Int32 )

# Create the MV folder specification with vector payload mode
folderSpec = cool.FolderSpecification( cool.FolderVersioning.SINGLE_VERSION,
                                       spec,
                                       cool.PayloadMode.VECTORPAYLOAD );
```

```
print 'creating folder'
f = db.createFolder( "/folder", folderSpec )

print 'filling folder'
f.setupStorageBuffer()
for c in range(0,2):
  for i in range(0,5):
      v = cool.IRecordVector()
      for j in range(0, i ):
          v.push_back( dataPtr(i,j,c) )
      f.storeObject( i, i+1, v , c )
f.flushStorageBuffer()

# Reading back data:
i = f.browseObjects( 0, 1000, cool.ChannelSelection.all() )
print "# of IOVs: %d" % i.size()
for obj in i:
  print "IOV [%d, %d[ c:%s " % ( obj.since(), obj.until(), obj.channelId() );
  for p in obj.payloadIterator():
    print "  %s" % p
```

-- MartinWache - 14-Oct-2010

Vector data in COOL: channels vs. vector payload vs. BLOBs

Vector data in COOL can be stored in a variety of different ways according to the specific needs of each user. Suppose for instance that a detector has 10k channels and one integer value of conditions data payload needs to be stored for each of them. In COOL there are at least three possibilities worth pointing out. Channels and inline BLOBs represent the opposite extermes, while vector payload sits somewhere in between.

1. **Channels**. Storing each of the 10k detector channels as a COOL channel will result in 10k rows in the COOL IOV table, including metadata (the IOV since/until) and data payload: metadata will be stored 10k times (on 10k rows) and data payload will be stored on those 10k rows too. This is a useful possibility if the integer value in each channel changes at different moments in time, but may result in a large overhead if all 10k detector channels are measured at the same time and logically share the same IOV since/until range, because the IOV since/until only needs to be stored once (and should be stored only once for consistency).
2. **Vector payload**. In this case a single IOV table row with a single IOV since/until metadata will be stored, referenced by 10k separate rows in the external table containing the vector payload. This reduces the overhead and data inflation in the system with respect to COOL channels. This also provides transparent access to the 10k integer values through the COOL API, possibly at the expense of performance as a complex SQL join between the IOV and external payload tables is required.
    - Vector payload in COOL has been introduced to replace the ATLAS CoraCool software layer. Its advantage with respect to the ATLAS CoraCool is that the complex joins between the COOL IOV table and the external vector payload table are performed at the SQL level in the database server behind COOL, rather than at the C++ level in the CoraCool client application.
3. **Inline BLOBs**. In this case a single IOV table with a single IOV since/until metadata and a single inline BLOB data payload will be stored. The 10k integer values are packed in the BLOB. The advantage of this solution is that it provides fast and compact access. The disadvantage is the lack of transparent access through the COOL API: users are responsible for the code that does the packing/unpacking of the 10k integer values into/from the BLOB. If any external tools (e.g. ATLAS data browsers) need to disaply these data, a good solution might be to enforce a uniform convention at the experiment level (as is done for int64 encoding/decoding into/from time stamps) and centrally develop an experiment-specific piece of code that should be used by all tools that should access these data.

-- AndreaValassi - 10-Jul-2013

This topic: Persistency > CoolVectorPayload
Topic revision: r2 - 2013-07-10 - AndreaValassi