ZODB Tips and Tricks

Presented at the 2005 Plone Symposium

Chris McDonough
Independent Consultant
Fredericksburg, VA
chrism@plope.com
http://www.plope.com

What Is ZODB

- Persistent Object Store
- Python is the indexing and query language
- Came from BoboPOS, which was written for Principia.
- Jim Fulton is principal author.
- Tim Peters is now the defacto maintainer.
- Collection of libraries that can be used outside Zope entirely

How Zope Uses ZODB

- Zope puts an “application” object in the root node of the ZODB.
- The “application” object is the root of “Zope space”.
- URL “traversal” in Zope begins at this object (this object is represented by '/')

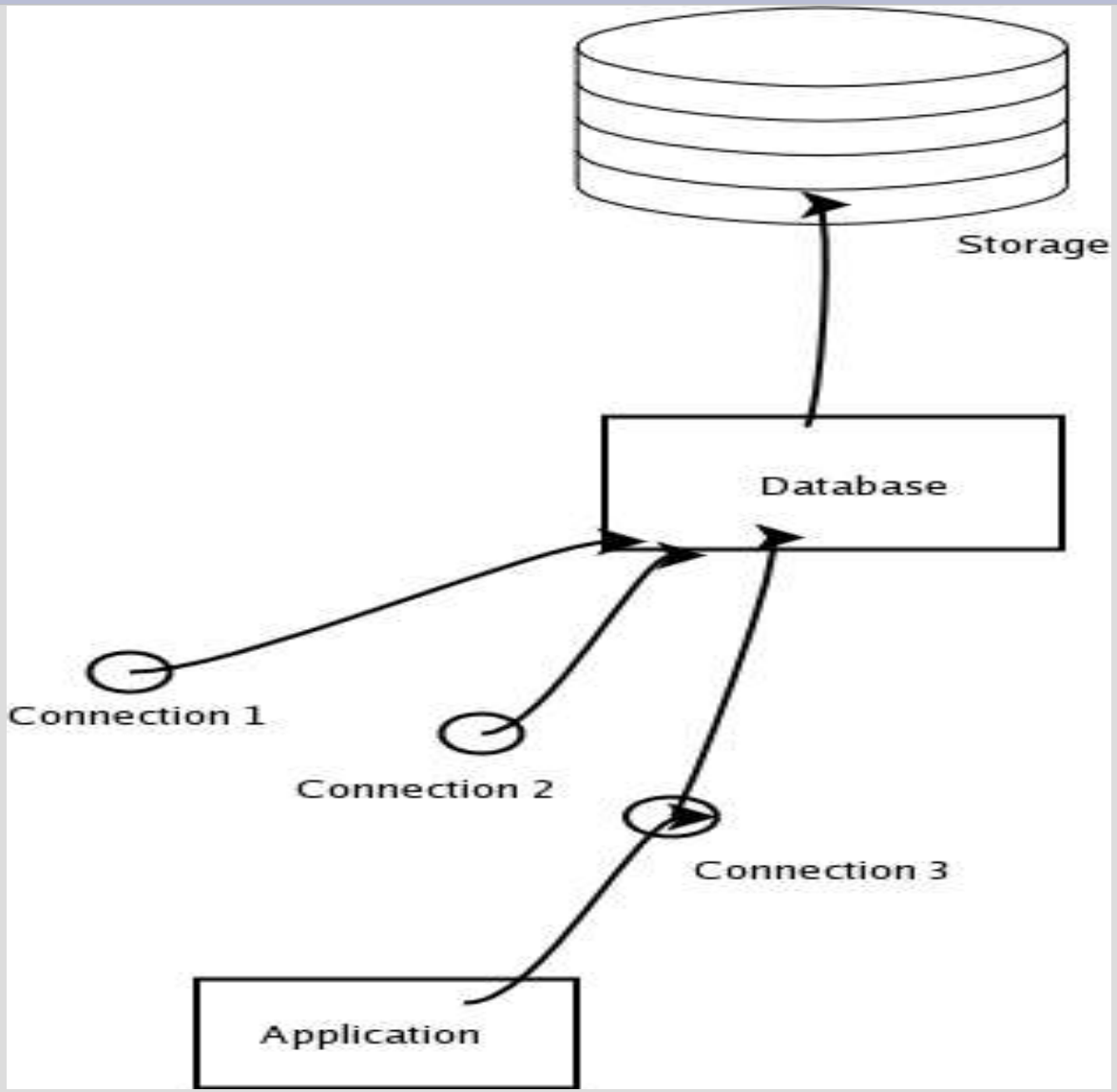
ZODB Releases

- ZODB can be used without Zope.
- Tim Peters creates new ZODB releases, basically timed with Zope releases.
- Find out how to download these ZODB releases from http://www.zope.org/Wikis/ZODB/FrontPage
- ZODB 3.2 == Zope 2.7
- ZODB 3.4 == Zope 2.8 / 3.0
- ZODB 3.5 == Zope 2.9 / 3.X

Using ZODB Without Zope

- **Zope allows you to use ZODB without knowing all of these concepts.**
- **Concepts**:
  - **Storage**: Writes bytes representing objects to disk. Ex: FileStorage, ClientStorage.
  - **Database**: Represents a pool of connections.
  - **Connection**: Provides application code with an interface to obtain the root object.
  - **Transaction**: a grouping of changes to persistent data.

How Applications Interact With ZODB



Persistent Objects

- ZODB uses the Python **pickle** module to serialize data. Any type of object that can be pickled can be stored in ZODB.
- Most builtin Python types can be pickled. Notable exceptions to that rule include file objects and function objects.
- To give clues to ZODB about when data is changed, **class instances** that are stored in ZODB must inherit from the **Persistent** class.

Persistent Instances

- Instances of classes that do not inherit from Persistent can still be stored in ZODB. However, if a class instance that does not inherit from **Persistent** is stored in ZODB, it effectively becomes a “write-once” object. Modifications to the object are not picked up by ZODB at commit time.
- Instances of any class that inherits from the **Persistent** base class are conventionally said to be “persistent objects”

Using ZODB Directly

- Examples follow
- These examples were developed using the ZODB release in a recent checkout of Zope 3. Earlier releases operate differently.
- “zopectl debug” is a great way to try out using ZODB from the Python command prompt.

Using ZODB Directly (example)

We can create a storage and a database, and store a string in the database.

```
$ python
>>> from ZODB.FileStorage import FileStorage
>>> from ZODB.DB import DB
>>> import transaction
>>> storage = FileStorage('My.fs')
>>> db = DB(storage)
>>> connection = db.open()
>>> root = connection.root()
>>> # we can begin a transaction now
>>> T = transaction.begin()
>>> # root is a dictionary-like object.
>>> root.keys()
[]
>>> root['myphrase'] = 'Hello World!'
>>> root.items()
[('myphrase', 'Hello World!')]
```

Using ZODB Directly (ex cont'd)

<cont'd session from previous slide>. If we abort the current transaction, our change goes away.

```
>>> transaction.abort()
>>> root.items()
[]
```

Using ZODB Directly (ex cont'd)

<cont'd session from previous slide>. We can create a persistent instance and store it in the database:

```
>>> from persistent.mapping import PersistentMapping
>>> T = transaction.begin()
>>> foo = PersistentMapping()
>>> root['myobject']= foo
```

Using ZODB Directly (ex cont'd)

<cont'd session from previous slide>. If we commit the current transaction and reopen the database, we can see that our changes have "stuck".

```
>>> transaction.commit()
>>> connection.close()
>>> connection = db.open()
>>> root = connection.root()
>>> root.keys()
['myobject']
>>> root['myobject'].__class__.__name__
'PersistentMapping'
```

Object Graphs

- When you attach an attribute to a persistent object via “setattr” (a.b = 'c') you are creating a persistent object graph.
- Data in any ZODB can be used as an object graph.
- Kind of like a table structure.
- Python is the language you can use to “query” this object graph. Compare this to a tradtional relational database, where you usually query your data using SQL.

Indexing

- Relational databases can use indexes to speed up queries.
- ZODB has no native indexing mechanism.
- **You** do the indexing of your data.
- The most popular indexing implementation is Zcatalog.
- Zope 3's catalog pieces are reportedly much easier to use outside of the rest of the Zope framework than are Zope 2's.

ZODB Strengths

- Pure Python (with some C extensions), so easy to install and use in Python applications.
- 99% “transparent” serialization of Python data structures. To store your data, you don't need to write SQL or create tables where you declare types.
- Lends itself naturally to hierarchical data stores. You just make a hierarchy of Python objects.

ZODB Weaknesses

- Python-only. Hard to tell people unfamiliar with Python how to get data out of it.
- Indexing is an application, not an intrinsic part of the database.
- Optimistic concurrency: conflict errors instead of locks.
- Flexibility. ZODB will allow you to shoot yourself in the feet and nobody will apologize afterwards.
- Not very efficient at storing large binary content (most databases aren't).

Recent Improvements to ZODB

- ZODB 3.3+ (Zope 2.8+) has many improvements over older versions:
  - Multiversion concurrency control (MVCC). Eliminates certain classes of performance-sapping conflict errors.
  - Can now persist “new-style” classes (classes which inherit from **object**).
  - 3.4+ has “savepoints” which are sort of like mini-transactions.

Coming Improvements to ZODB

- Blobs
  - *ctheune-blobsupport-branch* in SVN has code which makes “blobs” (large binary objects) a standard part of ZODB. More efficient than storing data in pickles. Slated to coincide with the release of Zope 2.9.
- (Zope-only) ZODB connection policies
  - Multiple connection pools. Zope decides which pool to use based on, perhaps, request parameters like user-agent (spiders, etc).

Improving ZODB Performance Now

- Check your ZODB database *cache-size* setting in **zope.conf** or **zeo.conf**. How big is big enough? As big as you can make it without causing your machine to swap.
- There is one ZODB cache per *connection* (ergo, Zope has more than one cache).
- If you use ZEO, experiment with the ZEO ClientStorage's *cache-size* setting too.
- Try **DirectoryStorage**, which doesn't keep a large index mapping in RAM.

Improving ZODB Performance Now

- If you use ZEO with persistent cache files, you may want to try tweaking the server's *invalidation-queue-size* if the connection between your ZEO server and your clients is less than reliable or if you frequently take your clients up and down.

Improving ZODB Performance Now (cont'd)

- Actually not many knobs to tweak in ZODB that effect performance other than cache sizes.
- The most important optimization you can perform is to write efficient code.
- Unfortunately, this is also the hardest way to optimize, because you need to manage all the details.
- No silver bullet, sorry.

Specific Coding Recommendations

- When writing code that accesses many persistent objects serially during the course of a single transaction, try calling the *cacheMinimize* on the connection object every, N iterations. Has potential to reduce memory usage.

Specific Coding Recommendations (cont'd)

- Use efficient data structures:
  - Learn about Btrees. If you have a lot of data in a Python dictionary or list, consider using a Btree or TreeSet instead.
  - Don't store large strings as single attributes of a persistent object. Break large strings up across many separate persistent objects or use a “blob” implementation to push large strings out to the filesystem as files.

Specific Coding Recommendations (cont'd)

- Avoid concurrent writes to the same persistent objects when possible. Counters are the canonical example of this but high-volume concurrent editing of content has the same effect.
- Use a relational database for what it's good for (search speed “cheaper”, system agnosticism, endless tuning knobs).

How to Get Involved

- Subscribe to the *zodb-dev@zope.org* mailing list.
- Report bugs and feature requests to the Zope “collector” using the topic “Database”.
- Be nice to Tim.
- Hire ZODB developers to get you the features you want.

Fin

THANKS!