Estrategias de Persistencia

ZODB

Integrantes:

- Cristian Suarez
- Ronny De Jesus
- Cristian Marchionne

ZOPE

Herramientas Utilizadas

- Utilizamos ZODB v3.6
- Compatible con Python v2.4
- Eclipse con PyDev (plugin para python)

ZOPE

Introduccion a ZODB

- ¿Que es ZODB?
- Propuestas de ZODB
- Arquitectura del Software
- Sistema de Almacenamiento

ZOPE

¿Que es ZODB?

- Es un sistema de persistencia para Python
- Es una base de objetos
- Persiste los datos en archivos .fs

Propuestas de ZODB

- Impacto minimo en el codigo existente de Python (transparente)
- Serializacion para grabar los objetos (pickle)
- Transacciones para controlar los cambios

Arquitectura del Software

- StandaloneZODB packages
  - Persistence, ZODB, ZEO
  - ExtensionClass, sundry utilities
- ZODB contiene
  - DB, Connection
  - Varios storage

Sistema de Almacenamiento

- Storage
  - Forma de manejar la persistencia en disco
- ZEO
  - Forma de almacenar la informacion distribuida

Conceptos claves de ZODB

- Persistencia por reachability
- Transacciones
- Manejo de recursos
  - Concurrencia
  - Memoria y cache

Persistencia por Reachability

- Todos los objetos son alcanzables desde el “root” almacenados en la DB
- Cualquier colaborador de un objeto persistente tambien es persistente
- Se pueden tener colaboradores no persistentes (volatiles)

Colaboradores Volatiles

- Colaboradores que no son persistidos llevan el prefijo ***_v_***

```
class F(Persistent):
def __init__(self, filename):
self._v_fp = open(filename)
>>> root['files'] = F('/etc/passwd')
>>> get_transaction().commit()
# later…
>>> root['files'].__dict__
{}
```

Transacciones

```
def beginTransaction(self):
    transaction.begin()
def comit(self):
    transaction.commit()
def rollback(self):
     transaction.abort()
```

ZOPE

Subtransaccion

- Se pueden crear subtransaciones dentro de una transaccion principal
  - Posee commit & abort individuales
  - Es comitteada realmente cuando la transacción principal lo hace.

Estado de los objetos

- Objectos en memoria
  - Cuatro estados
    - Unsaved
    - Up-to-date
    - Changed
    - Ghost

Cache de coneccion

- Cada conection tiene su propia cache
  - Los Objectos son referenciados por su OID
  - Los Objetos pueden ser “Ghost”
- Todas las cargas pasan por la cache
  - La Cache accede a los objectos recientes
  - Previene multiples copias de un objecto

Versionado

- Las transacciones pueden ser versionadas
- Si se cambia un objeto en una version, todos los cambios ocurren en la misma version hasta que:
  - se comitea
  - se aborta
- En caso contrario se produce un VersionLockError

Ejemplo de versionado

```
if db.supportsVersions():
db.open(version="myversion")

# Commit some changes, then
db.commitVersion("myversion")
# … or …
db.abortVersion("myversion")
```

Manejo de recursos

- Threads
  - Se pueden compartir el Storage y la DB
  - Las conecciones son individuales
  - Un thread por transaccion
- Memoria
  - Objetos levantados en memoria
  - Hace Swap In & Out cuando no hay espacio en la memoria

Ejemplo de Uso

- Crear un Storage
- Crear una DB que usa el storage
- Crear una Unidad de Trabajo

```
self.storage = FileStorage.FileStorage('database/database.fs')
self.db = DB(self.storage)
self.threadLocal = threading.local();

def createUnitOfWork(self):
    self.threadLocal.unitOfWork = UnitOfWork(self.db.open())
    return self.threadLocal.unitOfWork
```

Modulos Incluidos en ZODB

- PersistentMapping
- PersistentList
- BTrees

PersistentMapping

- Semantica similar a la de un Diccionario
- El metodo ***_p_changed*** se utiliza para avisar que hubo un cambio en el PersistentMapping

PersistentList

- Semantica similar a la de una Lista
- El metodo ***_p_changed*** se utiliza para avisar que hubo un cambio en el PersistentList

BTrees

- Diccionario implementado como Btree
  - Implementacion performante en C
- Difencias en el manejo de memoria
  - Los diccionarios almacenan todo en memoria
  - Los BTree son divididos en “buckets”
    - No estan todos en memoria al mismo tiempo

ACID

- El Framework respeta el concepto ACID

- Atomic
- Consistent
- Isolated
- Durable

Control Optimista de Concurrencia

- Niveles de isolacion
  - Locking: La transaccion lockea el objeto que usa
  - Optimistic: Aborta la transaccion conflictiva
- ZODB es optimistic
- Efecto en los estilos de programacion
  - Cualquier operacion puede producir ConflictError
  - Hay que manejar el error usando try/except

Utilizando ZEO

- Los Storages deberian ser abiertos con un solo proceso
  (aunque puedan ser multithreaded)
- ZEO permite abrir varios storage simultaneamente
- Los procesos pueden ser distribuidos en una red
- ZEO cache provee los datos como solo lectura si fallan los servidores

Migracion

- Los Storage pueden ser migrados mediante un protocolo iterator

```
src = FileStorage(“foo.fs”)
dst = Full(“BDB”) # Berkeley storage
dst.copyTransactionsFrom(src)
dst.close()
```

ZOPE

Referencias

BIBLIOGRAFIA

- **http://www.zodb.org**

ZOPE