

MongoDB + Pylons FTW:

Scalable Web apps with Python & NoSQL

- Niall O'Higgins



MongoDB ??

- Non-relational (NoSQL) document-oriented database
- Rich query language
- Flexible data model (JSON)
- Sharding and replication with automatic-failover
- CAP write quorum support



MongoDB vs MySQL vs BDB

- Feature Comparison

	MongoDB	MySQL	BDB
Master-Slave Replication	Yes	Yes	No
Automatic master failover	Yes	No	No
Ad-hoc Schemas	Yes	No	No
Manual index specification	Yes	Yes	Yes
Rich query language	Yes	Yes	No
Joins	No	Yes	No

MongoDB: NoSQL?


- Closer to MySQL than to BDB/Tyrant/Redis
- Less emphasis on Google-scale scalability *at this point* compared to e.g. Cassandra/HBase
- Today, MongoDB scales slightly better than MySQL IMHO.
- Main advantage over RDBMS is ease of development!



NoORM

- MongoDB document is a JSON object.
 - PyMongo driver exposes these as dictionaries
- E.g. `db.mycollection.insert(dict(myprop='foo!'))`
- You can even embed objects within a document:

E.g. `db.mycollection.insert(
dict(emails=
[dict(email_lower='niallo@pywebsf.org')]
))`



NoORM II

- You can query across properties, embedded or otherwise, easily.

E.g.

```
r = db.mycollection.find_one(  
    {emails.email_lower:'niallo@pywebsf.org'}  
)  
  
print r['emails'][0]['email_lower']
```



NoORM III

- SQL queries can get very complicated (huge joins, aggregate queries, etc)
- Marshalling SQL to/from objects is painful, tedious.
- Cross-RDBMS portability issues
- → ORMs can make sense with RDBMS
- At least in Python, I see absolutely no need for ORM with MongoDB.



More on querying MongoDB

- I alluded to a rich query language:
- \$lt, \$lte, \$gt, \$gte == <, <=, >, =>
- \$ne == <>, !=
- \$in == IN
- \$or == OR
- Sort() == ORDER BY
- Limit() == LIMIT
- Skip() == OFFSET
- Group() == GROUP BY



Geo Features

Geospatial querying out of the box:

- `$near (x, y)`: Return documents sorted by distance from point.
- `$within, $box/$center`: Return documents within bounds of a rectangle/circle. Earth is flat in 1.6.x and lower. Spherical model in 1.7.0 and up.
- Other limits: Only one geo index per collection. No sharding support.




More on querying MongoDB

- Sample rich query:


```
db.foo.find({'$or': [{'owner_account_id': Id},  
  {'shared_to': Id}], 'location': {'$near': [37.783,  
  -122.393]}}).skip(offset).limit(PAGE_SIZE)
```



MongoDB Gotchas

- Obviously not as mature as any RDBMS on the market. You can find bugs, but developers are extremely responsive.
 - 4MB limit per document.
 - Cap on number of indexes per collection.
 - No transactions, although some imperfect support for atomic operations. Race with e.g. Deleting single embedded object from list property.
 - → Not the right fit for every system!
 - → Think through your schema!
- 

Pylons

- Elegant, lightweight but fully-featured framework.
 - Perfect fit for mixed API and Web rendering environment we have at Catch.com
 - Threaded design, easy to program. No callbacks needed. Works fine with synchronous libraries.
 - Threads in Python suck due to the GIL.
 - Multi-core systems necessitate multi-process operation.
 - Options: Apache+mod_wsgi or multiple Pastes.
- 

mod_wsgi vs multiple Pasters

- Apache+mod_wsgi is kind of a pain to setup, configure.
- Past production experiences with mod_wsgi include memory leaks, CPU spinning.
- Paster is simple, built into Pylons. One Paster server maps cleanly to one Python interpreter process.
- LOLapps serve high-volume production traffic with Paster just fine.



Catch.com Paster

- Multiple Pastes per machine, each listening on a separate port.
- Nginx load balances between them all.
- Nginx also terminates SSL.
- No issues whatsoever handling the load today.



Paster Debugging Tip

- Configure `smtp_server`
- Point it at a Google Mail list
- Set up a Gmail label to filter messages into
- Now you have nicely searchable, real-time stack traces aggregated from each Paster. Very useful for debugging distributed systems!

