# Migrating from MySQL to MongoDB at Sluggy.com

Brendan W. McAdams

Evil Monkey Labs, LLC

Mongo Boston Conference - Sep. 19, 2010

# Outline



# Outline



# Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- On Advertising Revenues (e.g. Paying the Bills), Pete says: `I tend to think of advertising as a finky spastic mentally retarded cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh! food!"`
- Not Google... Not Trying to *be* Google. (Can't *afford* to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. Any downtime (be it bugs or system failure) means I get up at 3am to fix it.

# Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- On Advertising Revenues (e.g. Paying the Bills), Pete says: `I tend to think of advertising as a finky spastic mentally retarded cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh!  food!"`
- Not Google... Not Trying to *be* Google. (Can't *afford* to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. Any downtime (be it bugs or system failure) means I get up at 3am to fix it.

# Sluggy.com Rundown

- Live since August 25, 1997
- Updated Every Day (even if it's just filler)
- More Users == More Load
- More Load == More Hardware
- On Advertising Revenues (e.g. Paying the Bills), Pete says: `I tend to think of advertising as a finky spastic mentally retarded cat who sometimes wants to jump in my lap and other times wants to hiss at me and run for the litterbox and often walks in circles trying to figure out which of the two it wants, followed by dropping dead with a final thought..."ohhh!  food!"`
- Not Google... Not Trying to *be* Google. (Can't *afford* to think or scale like Google.
- No dedicated staff or operations budget - advertising revenues cover server costs. Any downtime (be it bugs or system failure) means I get up at 3am to fix it.

# Sluggy.com Rundown

## Some Stats

- 50GB/day (1.5TB of traffic/month on a single virtual box)
- 13 years of daily comics = 6500 image files (just for the comics)
- Artist is frequently late in updating. System has to handle random unexpected cache flushes & data updates.
- Erratic access behavior: Today's comic is always popular and related load can be easily mitigated. However, archives may also be hit heavily by new readers or links from a newer strip to previous storylines. Hard to expect where in 6500+ strips people may be digging from day to day.
- "LUMP" Stack (Lighttpd, Ubuntu, MongoDB & Python)

# Outline



# Sluggy.com Technology History

## 1997 - 2000

- **August 25, 1997**: Site Started, with basic code by Pete's friends/coworkers.
- Static HTML generated via midnight cron executing Perl.
- No dynamic content - Hand edited HTML for news, navigation, etc.
- File format requires globs:

```
000217a.gif
000217b.gif
000217c.gif
```

- All make up the panels for February 17, 2000. Artist likes & understands this format. Code looks for `yyMMdd*.(gif|jpg)` via glob and organizes them in order.
- *2000* - Original Developers split off and formed KeenSpot.com using same code & navigation concepts.

# Sluggy.com Technology History

## 2002

- Rewrite using MySQL & PHP (original goal: “No More HTML Editing”)
- Scope Creep == New Features. Dynamic headlines, news, predefined templates, dynamic navigation and a “Members Only Club”.
- *First Folly* - Reading MySQL and dynamically generating on each page request; Running dynamic code for essentially static content == *FAIL*. Disk I/O DoSing ... “call datacenter and cross fingers”
- Moved to smarty template caching, generating on-disk cache file upon first request (expires at midnight).
- Next 4 years became hellish with frequent Midnight crashes/failures as readers pound server for new comic.

# Sluggy.com Technology History I

## 2006

- Server move (For cost reasons) introduced new architecture problems
- Perceived cost savings pushed a move from SCSI/SAS disks to SATA
  - Between template files & comic file globs. Disks couldn't keep up.
  - Implemented **memcached** to cache templates off-disk, in memory. Cached glob results (but not files). Cached anything else not likely to change - expiry set to a week (midnight for "index")
  - Sessions performed poorly in both disk and MySQL - caching in memcached helped.
- Apache began crushing memory & disk I/O.
  - PHP isn't thread safe; requires forked Apache workers (children are *EXPENSIVE*)

# Sluggy.com Technology History II

## 2006

- Migrated to Lighttpd + FastCGI - IO & RAM usage of webserver & PHP became negligible (Lots of tweaking of handling of static files esp. stat caching w/ FAM & good event handling):

```
# without raising default fds (1024) we hit a lot of out of fds @ midnight
# File descriptors include network connections *AND* filesystem handles...
server.max-fds = 4096
# Use a well tuned event handler for connection handling
server.event-handler = "linux-sysepoll"
# Don't shred the disk pulling static files; w/o a custom engine
# like fam stat() runs for *EVERY* static file fetch.
server.stat-cache-engine = "fam"
# Severely limiting keep-alives paired w/ good Expires headers
server.max-keep-alive-requests = 4
server.max-keep-alive-idle = 2
# Ask politely that browsers don't keep redownloading static content
expire.url = (
  "/javascripts/" => "access 2 weeks",
  "/stylesheets/" => "access 2 weeks",
  "/icons/" => "access 2 weeks",
  "/images" => "access 1 weeks",
  "/images/comics/" => "access 1 days"
)
```

# Sluggy.com Technology History

## 2009

- Rewrote the system in Pylons (Python + SQLAlchemy[MySQL])
- Integrated **Beaker** caching decorators (templates & code blocks) - simplified adding caching code at need.
- Clean ORM model, light & fast with lots of caching.
- Ran significantly better than on PHP - infinitely more tunable, sensible, and sane (Not necessarily a knock on PHP - but it was a 10 year old codebase).
- **memcached** continued to become a big, rickety crutch (cascading failure sucks)

# Sluggy.com Technology History

## Aug. 2009

- **August 3, 2009**: Pylons system (v1.0) Live with MySQL backend.
- Huge amounts of our code (as much as 80%) was dedicated to converting UI Objects to and from Database objects. **WTF?**... Most initial bugs occurred in this model <-> view layer.
- No more forking - Pylons & Python run threaded via SCGI (Similar to FastCGI). System resources *significantly* less taxed by the presentation stack.

# Sluggy.com Technology History

## Aug. 2009

- **August 14, 2009**: v1.10 Went Live - MySQL replaced by MongoDB (and MongoKit)
- Easy Migration - MongoKit was quickly dropped in place and queries adjusted to new model (Stuck to MySQL schema as much as possible)
- Maintained all bug fixes on MySQL branch for a few weeks “just in case”
- Performance *vastly* improved.
- Over next few months, built tools to use MongoDB in place of **memcached** for caching (**mongodb_beaker**)
- *LAMP* replaced by **LUMP** (Lighttpd, Ubuntu, MongoDB & Python)
- A few things left in **memcached** through a combination of “makes sense there” and indolence.

# Sluggy.com Technology History

Sept. 2009

- MongoDB completely obviated dependency on dedicated Physical hardware; when a major issue with our ISP came up, migrated to Virtual hosting (*slicehost*) instead of Physical Hardware.
- Average system load is *0.05* on a 2G slice.
- MongoDB uses *1%* of CPU on average.
- Switchover to MongoDB version took 2 minutes (ran data conversion script, deployed new code tag, bounced webserver / Pylons app)
- No downtime in any way attributable to MongoDB since go live - now live with MongoDB over a year.

# Outline



# memcached can rapidly become a crutch. . .

## meant to make up for RDBMS' shortfalls but often masks other issues. . .

- **memcached** can be great for things you can afford to lose.
- It's not just about what you "can't afford to lose". Beware of cascading failures.
- Over reliance can cause self-DoSing after a crash, reboot, accidental flush (even of just one keyset) etc... Lesson learned the hard way.
- See *Coders at Work* (Siebel) for a great discussion with its creator, Brian Fitzpatrick (founded LiveJournal & now a Google employee), about what led to **memcached**'s creation.

# As long as I'm being hyperbolic. . . I

## MongoDB is a bionic leg replacement. . .

- MongoDB's MMAP system gives you a "free" MRU cache. Done right and simple; caching on MongoDB is durable, light and fast. Significantly educes amount of scalability-glue code.
- No piles of special code manage caching; if it falls out of memory cache, it is still safely persisted to disk.
- The more you can put in memory, the less you beat on your disks. Especially important on virtual hosting: Be a Good Neighbor.
- But. . . *don't* build your MongoDB system like a MySQL system (it'll work, but you rapidly lose speed and flexibility)

# As long as I'm being hyperbolic... II

## MongoDB is a bionic leg replacement...

- DBRefs should be used sparingly - favor embedded objects (don't be afraid to denormalize and duplicate data); autorefs can be even worse as there's a performance penalty imposed.
- Flexible schemas are good.
- Wasting your time mapping data back and forth between your presentation layer & RDBMS is not just tedious - it's error prone.
- Object Mappers for MongoDB are fantastic tools but don't overuse them - you take a huge flexibility & performance hit.
- Use field specifications, query operators, and atomic updates for maximum effectiveness. MongoDB excels at slicing out specific parts of a document - especially from embedded/nested fields.

# Caveats

- While there is a lot “wrong” with our first pass implementation, MongoDB has been consistent, performant and most importantly: *forgiving*.
- *Someone* has to enforce a consistent schema - if it’s not your datastore (like a RDBMS does) then your code or ops people (or both) have to.
- The MongoDB community is vibrant, supportive and consistently brilliant. *Use* your community to build the best possible product.
- Corollary: If there is not a vibrant, supportive and intelligent community behind a product you are evaluating. . . run.
- *Participate*: A community that takes and never gives back cannot thrive. Sharing your knowledge and experience goes a long way.

# Outline



# Open Source Code I

## mongodb_beaker: Beaker Caching for MongoDB

- Open Source caching plugin for the Python Beaker stack.
- Uses distutils plugin entry points.
- Switching from **memcached** to Beaker + MongoDB required a 2 line config file change:

```
- beaker.session.type = libmemcached
- beaker.session.url = 127.0.0.1:11211
+ beaker.session.type = mongodb
+ beaker.session.url = mongodb://localhost:27017/emergencyPants#sessions
```

- Lots of useful options in MongoDB Beaker.
- A few limitations on the beaker side which need changes in Beaker (manipulable cache data).
- Patch incubating for better replica, shard and master/slave support.

# Open Source Code I

## MongoKit-Pylons: Pylons patches for Python MongORM

- Added support to MongoKit to run within a Pylons environment (threadlocal setup of connection pool)
- Adding a globally available thread safe connection pool to Pylon was simple. Add 2 lines to config/environment.py:

```
from mongokit.ext.pylons_env import MongoPylonsEnv
MongoPylonsEnv.init_mongo()
```

# Open Source Code II

## MongoKit-Pylons: Pylons patches for Python MongORM

- Added a few other features to simplify SQLAlchemy migration
  - *setattr* / *getattr* support to allow **mongoDoc.*field*** instead of the dict interface (**mongoDoc['*field*']**)
  - DB Authentication
  - A few missing corners such as additional datatypes, enhanced index definitions on-document, group statement shortcuts, etc.
  - Integrated support for autoreferences (which was/are mostly a *very* bad idea)
- Changes merged into MongoKit Trunk (MongoKit has stellar unit test coverage)

# Outline



# Why MongoDB?

- Dynamic Querying
- Flexibility (embedded documents, atomic updates, query operators and server-side javascript.)
- CouchDB's approach appeared obtuse and rather unPythonic (not an indictment of CouchDB, simply reflective of my knowledge & opinion at the time)
- Tools like MongoKit allowed for easy replacement of existing MySQL ORM code with something almost identical
- *FAST*
- Great Support & Community Available.
  - Core developers/founders have serious experience in real scalability, and are active in community.
  - MongoDB Mailing List
  - IRC: freenode.net #mongodb

# Outline



# MySQL Schema Snippets I

## The Admin Table...

```
CREATE TABLE `admin_users` (
  `id` int(11) unsigned NOT NULL auto_increment,
  `username` varchar(45) collate latin1_general_ci NOT NULL default '',
  `password` char(32) collate latin1_general_ci NOT NULL,
  `display_name` varchar(64) collate latin1_general_ci default NULL,
  `email` varchar(255) collate latin1_general_ci NOT NULL default '',
  `avatar` varchar(255) collate latin1_general_ci default NULL,
  `last_ip` int(10) unsigned default NULL,
  `last_login_date` timestamp NOT NULL default '0000-00-00 00:00:00',
  `disabled` tinyint(1) default '0',
  PRIMARY KEY  (`id`),
  UNIQUE KEY `UNIQUE` (`username`),
  UNIQUE KEY `admin_users_uniq` (`email`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci
    PACK_KEYS=1;
```

# MySQL Schema Snippets I

## The News Table...

```
CREATE TABLE `news` (
  `id` int(11) unsigned NOT NULL auto_increment,
  `author_id` int(11) unsigned NOT NULL,
  `start_date` date NOT NULL,
  `end_date` date default NULL,
  `headline` varchar(255) NOT NULL,
  `story` text NOT NULL,
  `archive` tinyint(1) default '0',
  PRIMARY KEY  (`id`),
  KEY `news_author` (`author_id`),
  CONSTRAINT `news_author` FOREIGN KEY (`author_id`) REFERENCES `admin_users` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=201 DEFAULT CHARSET=latin1;
```

# SQLAlchemy Model I

## The Admin Object

```
class AdminUser(ORMObject):
    pass

t_admin_users = Table('admin_users', meta.metadata,
    Column('id', Integer, primary_key=True),
    Column('username', Unicode(45), nullable=False, unique=True),
    Column('password', Unicode(32), nullable=False),
    Column('display_name', Unicode(64)), # Should be unique?
    Column('email', Unicode(255), nullable=False, unique=True),
    Column('last_ip', IPV4Address, nullable=True),
    Column('last_login_date', MSTimeStamp, nullable=False),
    Column('avatar', Unicode(255), nullable=True),
    Column('disabled', Boolean, default=False),
    mysql_engine='InnoDB'
)

mapper(AdminUser, t_admin_users)
```

# SQLAlchemy Model I

## The News Object

```
class NewsStory(ORMObject):
    pass

t_news = Table('news', meta.metadata,
    Column('id', Integer, primary_key=True),
    Column('author_id', Integer, ForeignKey('admin_users.id'), nullable=False),
    Column('start_date', Date, nullable=False),
    Column('end_date', Date),
    Column('headline', Unicode(255), nullable=False),
    Column('story', Unicode, nullable=False),
    Column('archive', Boolean, default=False),
    mysql_engine='InnoDB'
)

mapper(NewsStory, t_news, properties={
    'author': relation(AdminUser, backref='news_stories')
})
```

# SQLAlchemy Model I

## Paypal: OH THE HORROR

```
class PaypalIPN(ORMObject):
    pass


t_paypal_ipn = Table('paypal_ipn', meta.metadata,
    Column('id', Integer, primary_key=True),
    Column('first_name', Unicode(64)),
    Column('last_name', Unicode(64)),
    Column('payer_business_name', Unicode(127)),
    Column('payer_email', Unicode(127)),
    Column('payer_id', Unicode(13)),
    Column('payer_status', Unicode(10)),
    Column('residence_country', Unicode(2)),
    Column('business', Unicode(127)),
    Column('receiver_email', Unicode(127)),
    Column('receiver_id', Unicode(13)),
    Column('item_name', Unicode(127)),
    Column('item_number', Unicode(127)),
    Column('quantity', Integer),
    Column('payment_date', Unicode(127)),
    Column('payment_status', Unicode(20)),
    Column('payment_type', Unicode(10)),
    Column('pending_reason', Unicode(20)),
    Column('reason_code', Unicode(20)),
    Column('mc_currency', Unicode(127)),
    Column('mc_fee', Unicode(127)),
    Column('mc_gross', Unicode(127)),
    Column('txn_id', Unicode(17)),
```

# SQLAlchemy Model II

## Paypal: OH THE HORROR

```
    Column('address_name', Unicode(255)),
    Column('address_street', Unicode(255)),
    Column('address_city', Unicode(255)),
    Column('address_state', Unicode(255)),
    Column('address_zip', Unicode(255)),
    Column('address_country', Unicode(255)),
    Column('address_country_code', Unicode(255)),
    Column('address_status', Unicode(15)),
    Column('notes', Unicode(255), nullable=True),
    mysql_engine='InnoDB'
)

mapper(PaypalIPN, t_paypal_ipn)
```

# MongoKit Model I

## Admin

```
class AdminUser(MongoPylonsDocument):
    use_autorefs = True
    use_dot_notation = True
    collection_name = 'admin_users'
    structure = {
        'username': unicode, # unique
        'password': unicode,
        'display_name': unicode,
        'email': unicode,
        'last_ip': unicode,
        'last_login_date': datetime.datetime,
        'avatar': unicode,
        'disabled': bool,
    }

    required_fields = ['username', 'password', 'email']
    default_values = {'disabled': False, 'last_login_date': datetime.datetime.now()}

    indexes = [
        {
         'fields': ['username', 'password'],
         'ttl': 86400
        },
        {
         'fields': ['password'],
         'ttl': 86400
        }
    ]
```

# MongoKit Model II

## Admin

# MongoKit Model I

## News

```
class NewsStory(MongoPylonsDocument):
    use_dot_notation = True
    use_autorefs = True
    collection_name = 'news'
    structure = {
        'author': AdminUser,
        'headline': unicode,
        'story': unicode,
        'start_date': datetime.datetime,
        'end_date': datetime.datetime,
        'archived': bool # default false
    }

    required_fields = ['author', 'headline', 'story', 'start_date']
    default_values = {
      'start_date': TODAY,
      'archived': False,
    }

    indexes = [
        {
            'fields': ['start_date', 'end_date'],
            'ttl': 86400,
        },
        {
            'fields': 'archived',
            'ttl': 86400,
        }
```

# MongoKit Model II

## News

```
    ]
```

# MongoKit Model

## PayPal: OH THE. . . that's not so bad.

# MongoKit Model I

## PayPal: “Instead of using the ORM. . . ”

```
ipn_db = None
try:
    dbh = MongoPylonsEnv.mongo_conn()\
        [MongoPylonsEnv.get_default_db()]['defenders_paypal_ipn']
    ipn_id = dbh.insert(dict(request.POST.items()), safe=True)
    ipn_db = dbh.find_one({'_id': ipn_id})
    if not ipn_db:
        raise Exception('could not lookup, post-insert')
except Exception, e:
    log.exception("Paypal IPN Error: Unable to commit IPN data to "
                  " Database: %s " % repr(e))
    raise DefendersIPNException(repr(e))
```

# MySQL -> Mongo Migration I

## Admin Migration

```
db = mongoModel.AdminUser._get_connection()
conn = db.connection()
conn.drop_database('emergencyPants')

admins = {}
for user in meta.Session.query(AdminUser).all():
    _admin = mongoModel.AdminUser(doc={
        'username': user.username,
        'password': user.password,
        'avatar': user.avatar,
        'disabled': user.disabled,
        'display_name': user.display_name,
        'email': user.email,
        'last_ip': unicode(user.last_ip),
        'last_login_date': user.last_login_date,
    }).save()
    admins[user.id] = _admin

mongoModel.AdminUser.get_collection().ensure_index('password',
            direction=ASCENDING, unique=True)
mongoModel.AdminUser.get_collection().ensure_index(
            [('username', ASCENDING),
            ('password', ASCENDING)])
```

# MySQL -> Mongo Migration I

## News Migration

```
print "Importing News Stories."
for story in meta.Session.query(NewsStory).all():
    _story = mongoModel.NewsStory(doc={
        'author': admins[story.author_id],
        'headline': story.headline,
        'story': story.story,
        'start_date': convert_date(story.start_date),
        'end_date': convert_date(story.end_date),
        'archived': story.archive
    }).save()

print "Setting up news story indices."
mongoModel.NewsStory.get_collection().\
    ensure_index('archived', direction=ASCENDING)
mongoModel.NewsStory.get_collection().\
    ensure_index([('start_date', ASCENDING),
                  ('end_date', ASCENDING)])
```

# Looking for news I

## In Mongo. . .

- Before, with MySQL/SQLAlchemy:

```
def _get_news(date):
   news = meta.Session.query(NewsStory).\
      filter(and_(NewsStory.archive==False, NewsStory.start_date<=date,
             or_(NewsStory.end_date==None, NewsStory.end_date>date)))\
             .order_by(NewsStory.start_date.desc()).all()
    return news
```

- After, with MongoKit:

```
def _get_news(date):
    news = NewsStory.all({
        'archived': False,
        'start_date': {'$lte': c._today}
    }).where('this.end_date == null || this.end_date >= new Date()').\
       sort('start_date', -1)

    return news
```

# Questions?

## Contact Info

- Contact Me
  - twitter: **@rit**
  - email: **bwmcadams@gmail.com**
  - bitbucket: `http://hg.evilmonkeylabs.net` (For MongoKit & Beaker code)
  - github: `http://github.com/bwmcadams` (Where most of my newer code lives)
- Pressing Questions?
  - IRC - freenode.net **#mongodb**
  - MongoDB Users List - `http://groups.google.com/group/mongodb-user`
- Mongo Python Language Center - `http://api.mongodb.org/python/index.html` (Tutorial, API Docs and links to third party toolkits)