



mongoDB

NoSQL - What we've
learned with mongoddb

Paul Pedersen, Deputy CTO

paul@10gen.com

DAMA SF

December 15, 2011

DW2.0 and NoSQL

- management decision support
- integrated access
 - local v. global
 - structured v. unstructured
- data life-cycle
 - current v. recent v. archival
- high volume
- large size
- multi-dimensional data

mongodb design goals

- support agile development
 - rich data model, well matched to OO data
 - general-purpose DBMS
 - simple, but powerful query syntax
 - multiple secondary indexes, geo indexes
 - smooth path to horizontal scaling
 - broad integration into existing ecosystems

mongodb design goals

- **web scale** (CAP Theorem tradeoff)
 - eventual consistency: single master
 - availability: non-blocking writes, journal-blocking writes, replication-blocking writes
 - durability: journaling
 - partition: multi-data center replication
 - easy horizontal scaling by data sharding

Footnote: CAP Theorem

- Consistency (all nodes see the same data at the same time)
- Availability (every request receives a response about whether it was successful or failed)
- Partition tolerance (the system continues to operate despite arbitrary message loss)

According to the theorem, a distributed system can satisfy any two of these guarantees at the same time, but not all three.

mongodb architecture

- BSON object store, unique _id
- mmap memory management
- B-Tree indexing
- single-master replication
- logical key-range partition sharding
- global configuration manager

JSON / BSON

example

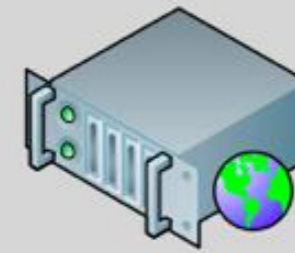
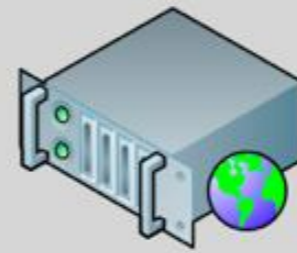
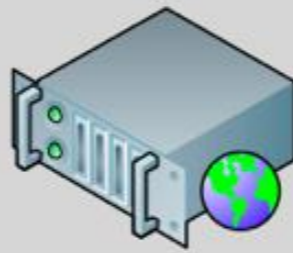
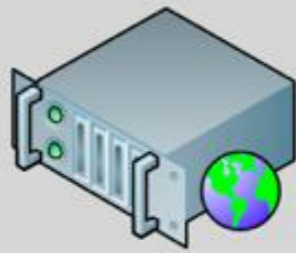
```
{  
  "_id" : 123456789,  
  "name" : { "first": "mongo", "last": "db" },  
  "address" : "555 University Ave., Palo Alto CA  
94301",  
  "checkins" : [ "starbucks", "ace hardware", "evvia" ],  
  "loc" : [38.57, 121.62 ]  
}
```

Types:

- unique id
- strings, dates, numbers, bool
- embedded documents
- arrays
- regex
- binary data

Application Tier

east data center



a1.acme.com:8080
s1.acme.com:27017

a2.acme.com:8080
s2.acme.com:27017

a3.acme.com:8080
s3.acme.com:27017

a4.acme.com:8080
s4.acme.com:27017

east data center

Replica Set A rs_a



e1.acme.com:27018
c1.acme.com:27019



e2.acme.com:27018



w1.acme.com:27018
c3.acme.com:27019

Replica Set B rs_b



e3.acme.com:27018



e4.acme.com:27018
c2.acme.com:27019



w2.acme.com:27018

Replica Set C rs_c



e5.acme.com:27018



e6.acme.com:27018



w3.acme.com:27018

west data center

Data Tier



mongodb scale

- What sort of 'big data' ?
 - all the text / images / location data / metadata driving your web app
 - where you've been
 - what you like
 - what you've said
 - what you bought
 - who liked / retweeted / linked to you

mongodb scale

- We are running instances on the order of:
 - 25B objects
 - 50TB storage
 - 50K qps per server
 - 500 servers

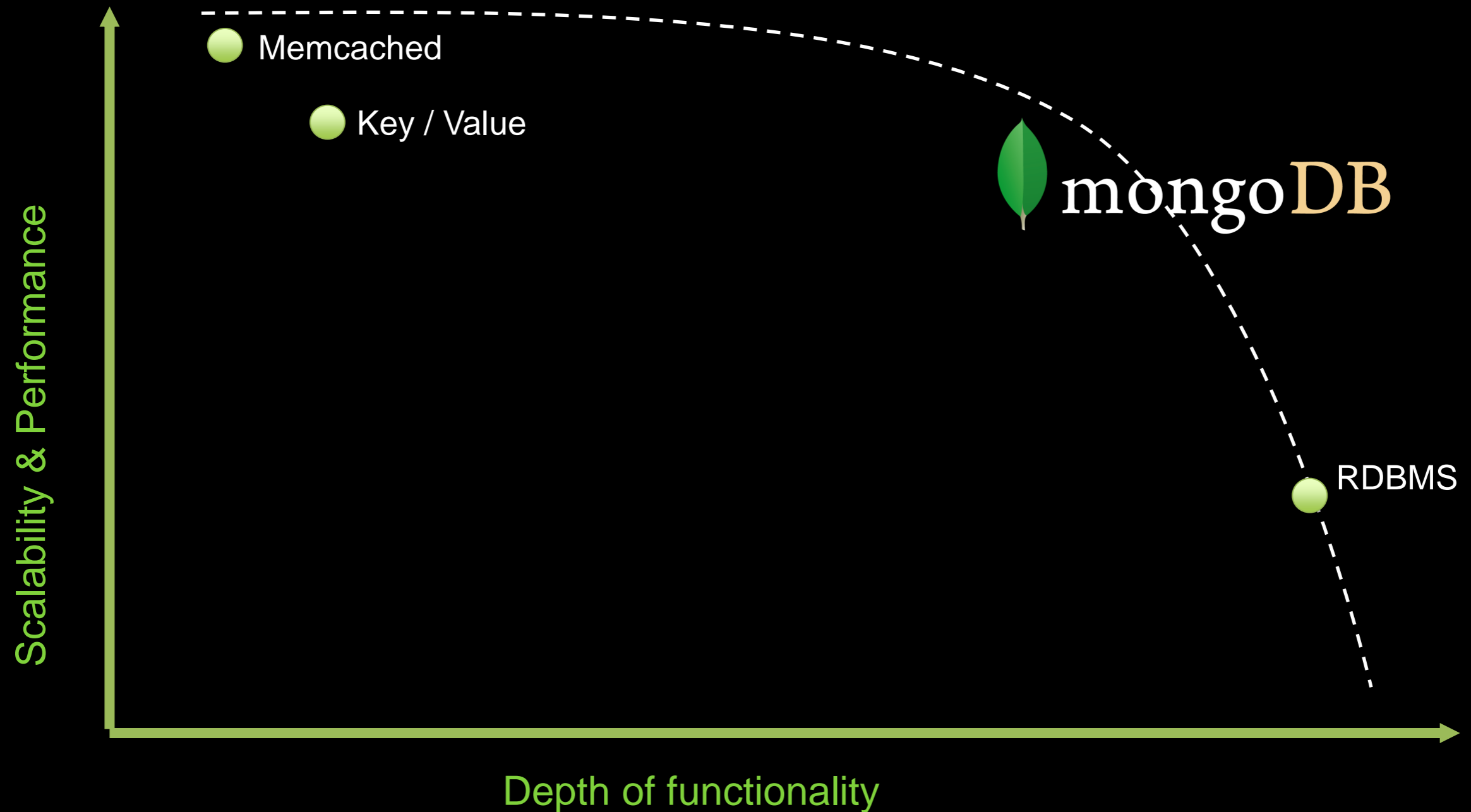
What we've learned

- Ease of use
 - schema flexibility
 - good match to OO data and processing
 - easy migration path:
- single instance
-  replica set
-  sharded replica sets

What we've learned

- users, developers want:
 - high availability
 - high query volumes
 - system monitoring / tracing
 - fast reboot / recovery
 - cross-data center replication
 - rich geo indexing
 - schema flexibility

Make it as simple as possible, but no simpler



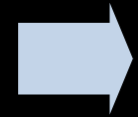
RDBMS systems



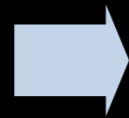
Costs go up

Productivity goes down

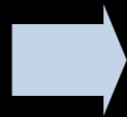
**Project
Start**



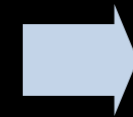
**De-
normalize
data model**



**Stop using
joins**



**Custom
caching
layer**



**Custom
sharding**

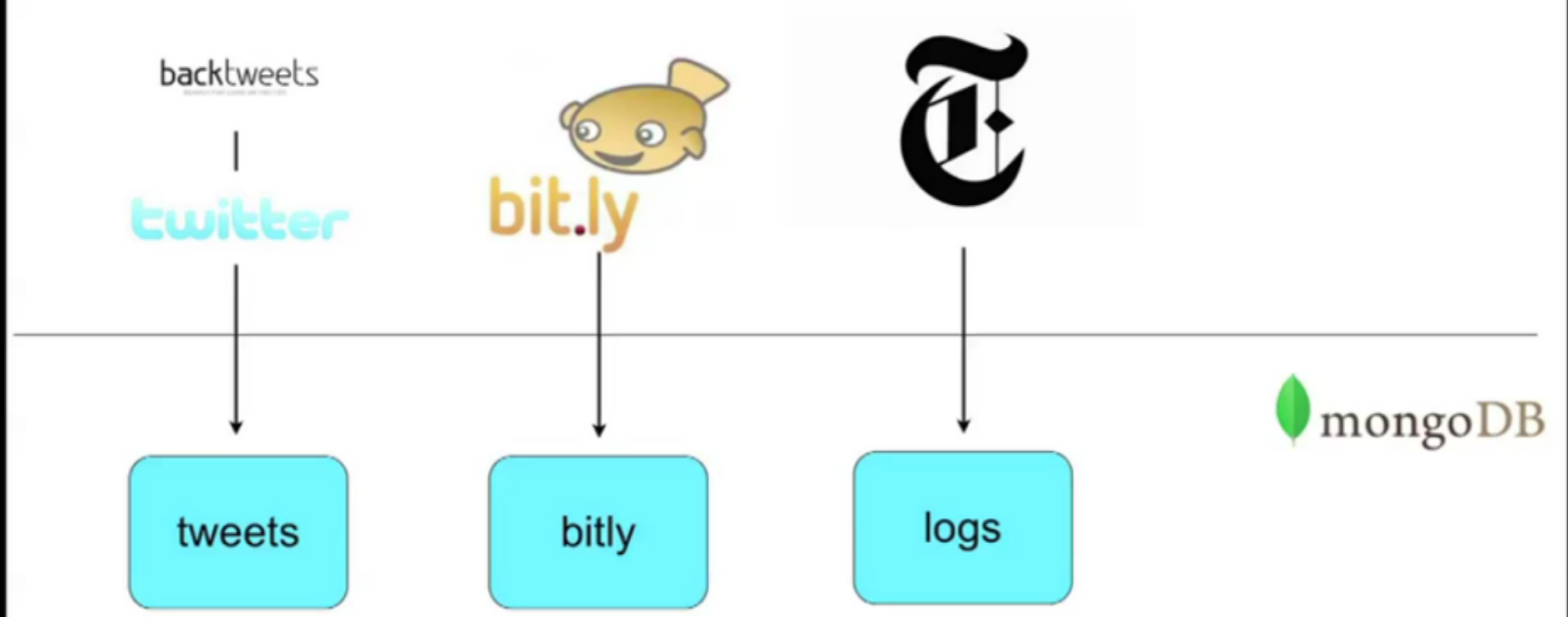
Case study: NYTimes R&D

- [Project cascade](#) - analyzes social media news sharing
 - data exploration tool: 600 articles/day
 - nytime click log, tweets (25K/day), retweets, bit.ly expansion requests (25K/day), 100GB/month
 - 'cascading' data sharing
 - polar charts, time radius = velocity of news sharing
 - influence = time series traffic spikes
 - (blue=click into nytimes.com page, red=tweet, blue=bit.ly compress, yellow=bit.ly uncompress)

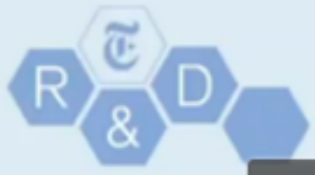
Case study: NYTimes

DDD

mongoDB and Cascade Architecture

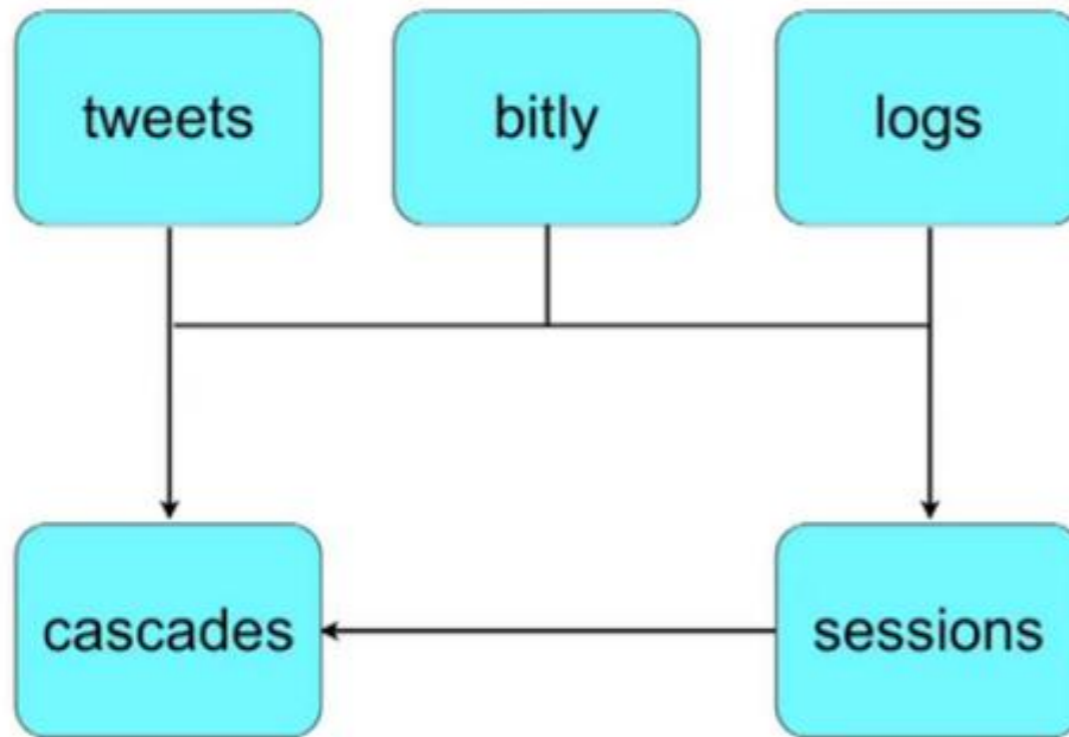
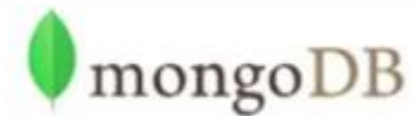


Why we love mongoDB: It eats this data up! We just dump data from our Python scripts or APIs into mongo and we're good to go.



Case study: NYTimes

mongoDB and Cascade Architecture



Raw tweets / bitly / logs are converted into site sessions and cascades.

Why we love mongoDB: Cascades are trees. Ever try to manage hierarchical data in a relational DB? With mongoDB we can easily create Cascade objects as embedded documents.

11



18:44

36:13



Case study: NYTimes

NOB

Summary

- Overall, mongoDB is great for us because:
 - We can't anticipate clients' data sizes - need scalable DB.
 - Schema-less approach is critical for flexibility during research.
 - Map/reduce is a nice framework for customizable batch operations.
 - We do a lot of data collection, which mongo takes in stride.
 - Geo-indexing is super useful for any location-based projects.



Case study: NYTimes R&D

Key benefits cited by nytimes:

- ease of expansion
- ease of use: dump json straight into the db, don't worry about the schema
- ease of use: in-place update, field insert
- language adapters: python, java, ruby
- map-reduce built in
- geo indexing
- multiple secondary indexes easy to use

Case study: foursquare

- location-based social network - “check-in” to bars, restaurants, museums, parks, etc
 - friend-finder (where are my friends right now?)
 - virtual game (badges, points, mayorships)
 - city guide (local, personalized recommendations)
 - location diary + stats engine (where was I a year ago?)
 - specials (get rewards at your favorite restaurant)

Case study: foursquare

- >9M users
- ~3M checkins/day
- >15M venues
- >300k merchants
- >60 employees

foursquare <3's mongodb

- fast
- indexes & rich queries
- sharding, auto-balancing
- replication (see: <http://engineering.foursquare.com/>)
- geo-indexes
- amazing support

Case study: foursquare

- benefits cited by foursquare:
 - fast
 - multiple secondary indexes
 - rich queries
 - sharding, auto-balancing
 - replication
 - geo-indexes
 - amazing support

Case study:

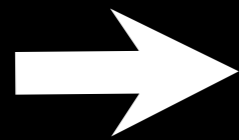
foursquare

mongodb: lessons learned

- keep working set in memory
- avoid long-running queries (reads or writes)
- monitor everything (especially per-collection stats)
- shard from day 1
- beware EBS
- use small field names for large collections

Data life-cycle management

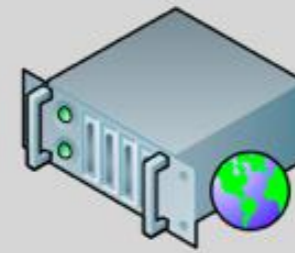
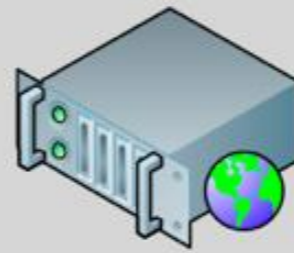
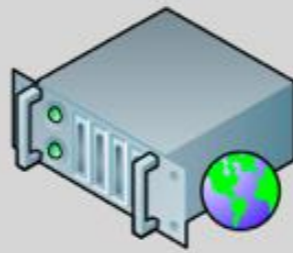
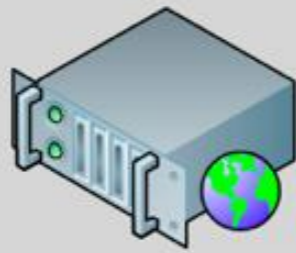
- Current
- Recent
- Archival



- TTL collections
- Date sharding

Application Tier

east data center



a1.acme.com:8080
s1.acme.com:27017

a2.acme.com:8080
s2.acme.com:27017

a3.acme.com:8080
s3.acme.com:27017

a4.acme.com:8080
s4.acme.com:27017

east data center

Replica Set A rs_a



e1.acme.com:27018
c1.acme.com:27019



e2.acme.com:27018



w1.acme.com:27018
c3.acme.com:27019

Replica Set B rs_b



e3.acme.com:27018



e4.acme.com:27018
c2.acme.com:27019



w2.acme.com:27018

Replica Set C rs_c



e5.acme.com:27018



e6.acme.com:27018



w3.acme.com:27018

west data center

Data Tier

Thanks

