NoSQL and MongoDB

Sandeep Parikh

Partner Technical Solutions, MongoDB Inc.
What is NoSQL?
The Basics

- Non-relational structure
- Beyond rows/columns
- Horizontal scalability
- Highly distributed
- Started out as “No SQL”
  - Lately that’s changed to “Not Only SQL”
“A NoSQL database provides a mechanism for storage and retrieval of data that use looser consistency models than traditional relational databases in order to achieve horizontal scaling and higher availability.”

http://en.wikipedia.org/wiki/NoSQL
Why is NoSQL popular?
What’s Driving Popularity?

• Data model flexibility
  – Matching the data to your applications

• Scale across many smaller systems
  – “Commodity hardware”

• Falling storage prices

• Better virtualization

• Increase in cloud computing
Why Do Big Companies Care?

- Transparency from open source
- Development practices are out in the open
- Licensing fees keep rising
- Adoption of more agile techniques
  - For development and deployment
Data Has Changed Over The Years

• **Volume**
  – Data generated and collected has grown drastically
  – Applications can generate 1GB of logs per day

• **Variety**
  – Evolving data and varying formats
  – Not always known at design time

• **Velocity**
  – Inbound data speed
  – Fast reads/writes across much larger datasets
What’s Wrong with NoSQL?
The Downsides

• No universal query language like SQL
  – Each technology does things differently

• SQL is very powerful and expressive

• Relational databases are very mature
  – 40+ years vs. 4+ years

• Relational databases are part of a vast ecosystem
  – Tools
  – Libraries
  – Integrations
NoSQL Overview

- Cassandra:
  - Key to multiple values, grouped in ColumnFamilies
  - Each key can have different columns
  - Dynamo operational model, BigTable data model

- HBase
  - Tightly integrated with Hadoop
  - BigTable implementation
NoSQL Overview

• Riak
  – Distributed key-value store
  – Dynamo implementation

• Couchbase
  – Memebase (memcached) plus some CouchDB features
  – Key-value store; values can be JSON objects
  – Create pre-computed views to query inside JSON objects
MongoDB
Goals

• Scale horizontally over commodity systems
• Incorporate what works for RDBMSs
  – Rich data models, ad-hoc queries, full indexes
• Move away from what doesn’t scale easily
  – Multi-row transactions, complex joins
• Use idomatic development APIs
• Match agile development and deployment workflows
Key Features

• Data stored as documents (JSON)
  – Dynamic-schema

• Full CRUD support (Create, Read, Update, Delete)
  – Ad-hoc queries: Equality, RegEx, Ranges, Geospatial
  – Atomic in-place updates

• Full secondary indexes
  – Unique, sparse, TTL

• Replication – redundancy, failover

• Sharding – partitioning for read/write scalability
MongoDB Drivers and Shell

Drivers
Drivers for most popular programming languages and frameworks

Shell
Command-line shell for interacting directly with database

```
> db.collection.insert({product: "MongoDB", type: "Document Database"})
>
> db.collection.findOne()
{
    "_id" : ObjectId("5106c1c2fc629bfe52792e86"),
    "product" : "MongoDB",
    "type" : "Document Database"
}
```
Document Oriented, Dynamic Schema

Relational

<table>
<thead>
<tr>
<th>Person:</th>
<th>Surname</th>
<th>First_Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Miller</td>
<td>Paul</td>
<td>London</td>
</tr>
<tr>
<td>1</td>
<td>Ortega</td>
<td>Alvaro</td>
<td>Valencia</td>
</tr>
<tr>
<td>2</td>
<td>Huber</td>
<td>Urs</td>
<td>Zurich</td>
</tr>
<tr>
<td>3</td>
<td>Blanc</td>
<td>Gaston</td>
<td>Paris</td>
</tr>
<tr>
<td>4</td>
<td>Bertulini</td>
<td>Fabrizio</td>
<td>Rom</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Car:</th>
<th>Model</th>
<th>Year</th>
<th>Value</th>
<th>Pers_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Bentley</td>
<td>1973</td>
<td>100000</td>
<td>0</td>
</tr>
<tr>
<td>102</td>
<td>Rolls Royce</td>
<td>1965</td>
<td>330000</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>Peugeot</td>
<td>1993</td>
<td>300</td>
<td>3</td>
</tr>
<tr>
<td>104</td>
<td>Ferrari</td>
<td>2005</td>
<td>150000</td>
<td>4</td>
</tr>
<tr>
<td>105</td>
<td>Renault</td>
<td>1998</td>
<td>2000</td>
<td>3</td>
</tr>
<tr>
<td>106</td>
<td>Renault</td>
<td>2001</td>
<td>700</td>
<td>3</td>
</tr>
<tr>
<td>107</td>
<td>Smart</td>
<td>1999</td>
<td>2000</td>
<td>2</td>
</tr>
</tbody>
</table>

MongoDB

```javascript
```
Working With MongoDB
> var new_entry = {
  firstname: "John",
  lastname: "Smith",
  age: 25,
  address: {
    street: "21 2nd Street",
    city: "New York",
    state: "NY",
    zipcode: 10021
  }
}

> db.addressBook.save(new_entry)
Querying

```javascript
> db.addressBook.find()
{
  _id: ObjectId("4c4ba5c0672c685e5e8aabf3"),
  firstname: "John",
  lastname: "Smith",
  age: 25,
  address: {
    street: "21 2nd Street", city: "New York",
    state: "NY", zipcode: 10021
  }
}

// _id is unique but can be anything you like
```
Indexes

// create an ascending index on “state”
> db.addressBook.ensureIndex({state:1})

> db.addressBook.find({state:”NY”})
{
    _id: ObjectId(“4c4ba5c0672c685e5e8aabf3”),
    firstname: “John”,
    ...
}

> db.addressBook.find({state:”NY”, zip: 10021})
Queries

// Query Operators:
// $all, $exists, $mod, $ne, $in, $nin, $nor, $or,
// $size, $type, $lt, $lte, $gt, $gte

// find contacts with any age
> db.addressBook.find({age: {$exists: true}})

// find entries matching a regular expression
> db.addressBook.find( {lastname: /^smi*/i } )

// count entries with "John"
> db.addressBook.find( {firstname: 'John'} ).count()
Updates

// Update operators
// $set, $unset, $inc, $push, $pushAll, $pull,
// $pullAll, $bit

> var new_phonenumber = {
  type: "mobile",
  number: "646-555-4567"
}

> db.addressBook.update({ _id: "..." }, {
  $push: {phonenumbers: new_phonenumber}
});
Nested Documents

{
    _id: ObjectId("4c4ba5c0672c685e5e8aabf3"),
    firstname: "John", lastname: "Smith",
    age: 25,
    address: {
        street: "21 2nd Street", city: "New York",
        state: "NY", zipcode: 10021
    }
    phonenumbers: [ {
        type: "mobile", number: "646-555-4567"
    } ]
}
Secondary Indexes

// Index nested documents
> db.addressBook.ensureIndex({"phonenumbers.type":1})

// Geospatial indexes, 2d or 2dsphere
> db.addressBook.ensureIndex({location: "2d"})
> db.addressBook.find({location: {$near: [22,42]}})

// Unique and Sparse indexes
> db.addressBook.ensureIndex({field:1}, {unique:true})
> db.addressBook.ensureIndex({field:1}, {sparse:true})
Additional Features

• Geospatial queries
  – Simple 2D plane
  – Or accounting for the surface of the earth (ellipsoid)

• Full Text Search

• Aggregation Framework
  – Similar to SQL GROUP BY operator

• Javascript MapReduce
  – Complex aggregation tasks
Open Source

• MongoDB source code is on Github
  – https://github.com/mongodb/mongo

• Issue tracking for MongoDB and drivers
  – http://jira.mongodb.org
Support

• Tickets are created by
  – Customer support
  – Community support (Google Groups, StackOverflow)
  – Community members
  – MongoDB employees

• Tickets can be voted on and watched to track progress

• Follow-the-Sun support

• All technical folks spend time doing community and customer support
Development

• Issues are triaged by CTO and engineering managers

• Then assigned into buckets, like
  – Specific version (ex. 2.7.1)
  – Desired version (ex. 2.7 desired)
  – Planning buckets
  – Unscheduled

• Engineers assign themselves tickets

• Once code is committed, a code review is needed
QA and Testing

- Code reviewer nominates for QA
- Unit tests are done by engineer
- Integration tests are done by QA team
- Support/Consulting/Architect teams do
  - Internal feature reviews/presentations
  - Beta testing with community and customers
- Documentation updates are linked to QA tickets
Questions?

• Sandeep Parikh
  – sap@mongodb.com
  – @crcsmnky

• MongoDB
  – MongoDB, drivers, documentation
    • http://www.mongodb.org
    • http://docs.mongodb.org
  – Free online training, presentations, whitepapers
    • http://www.mongodb.com