NoSQL the Modern MUMPS

Will LaForest
Senior Director, MongoDB Federal
will@mongodb.com
@wlaforest
Dawn of Databases to Present

- IMS (hierarchical) invented
- SQL invented
- Oracle founded
- Client Server
- Web applications
- SOA
- BigTable
- Cloud Computing
- NoSQL Movement
- MongoDB released

- 1965: IMS (hierarchical) invented
- 1970: Oracle founded
- 1975: Codd publishes relational model paper in 1970
- 1980: PC's gain traction
- 1985: 3 tier architecture
- 1990: WWW born
- 1995: SOA
- 2000: Cloud Computing
- 2005: NoSQL Movement
- 2010: MongoDB released
Why RDBMs?
## Relational Databases

<table>
<thead>
<tr>
<th>Attribute</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Tuple

<table>
<thead>
<tr>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

mongoDB
Relational Databases

Diagram:

- Category
- BlogCategory
- Author
- Blog
- BlogTag
- Content
- Tag
Relational Database Strengths

• Data stored in a RDBMS is very compact
  – duplication often normalized out
  – disk more expensive

• Separation of data management from app code

• Rigid schemas helps optimize joins and storage

• Rigid schemas but flexible adhoc queries

• Good general purpose operational data store

• Rich ecosystem
  – Huge adoption and people with skill sets
  – Huge selection of tools, libraries and integrations
MUMPS
Key MUMPS Aspects

• MUMPS is a language AND a database
• True global variables
  – Accesses like a normal variable but persistent and accessible across processes
• Hierarchical database that can be thought of as multidimensional sparse associative arrays
• SET ^Car("Door","Color")="BLUE"
• No data types (coercion used at runtime)
• Schema agnostic
Why MUMPS adoption

• Created out of the healthcare domain
• Open source drove adoption
• Early and deep penetration into the market

• Data model a better fit for the healthcare domain
• Performance was superior to RDBMS
NoSQL
Dawn of Databases to Present

- IMS (hierarchical) invented
- SQL invented
- Oracle founded
- Client Server architecture
- Web applications
- SOA
- BigTable
- Cloud Computing
- NoSQL Movement

- MUMPS
- PC's gain traction
- 3 tier architecture
- WWW born
- NoSQL Movement released

- Codd publishes relational model paper in 1970
- IDS (network)
- 1965
- 1970
- 1975
- 1980
- 1985
- 1990
- 1995
- 2000
- 2005
- 2010
Don’t Need All 3 Vs to be Big Data

Patient Records (volume, variety)

PGD/Crowd Sourcing (variety)

Asset Management (variety, velocity)

Cyber (volume, velocity, variety)
Why not the RDBMS?

- Checks the boxes for a general purpose operational data store
- But not the boxes for big data
- RDBMS Weakness
  - Need to know exactly what your data looks like a-priori
  - Scales vertically
- Both of these weaknesses map directly to all three components of big data
  - Variety means you don’t know what the schema is or it changes frequently
  - Must scale horizontally to handle velocity and volume
- Not great for modern agile development
Enter NoSQL Movement

• NoSQL movement driven by modern data challenges
• NoSQL = Not Only SQL
• Operational and on-line data
• Predominantly open source offerings
• But also built for big data
  – Scales horizontally
  – High performance
  – No or loose schema configured a-priori
Modern NoSQL Databases

• Key-Value Stores
  – Key-Value Stores
  – Value (Data) mapped to a key (think primary)

• Big Table Descendants
  – Looks like a distributed multi-dimension map
  – Data stored in a column oriented fashion
  – Predominantly hash based indexing

• Document oriented stores
  – Data stored as either JSON or XML documents
Key/Value

- Scalable but
- Very simple data model
- Limited indexing and querying capabilities

<table>
<thead>
<tr>
<th>“Will”</th>
<th>“<a href="mailto:will@10gen.com">will@10gen.com</a>”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Chris”</td>
<td>“<a href="mailto:chris@10gen.com">chris@10gen.com</a>”</td>
</tr>
<tr>
<td>Will-obj</td>
<td>[4e61 6d65 3a57 …]</td>
</tr>
</tbody>
</table>
Big Table Descendants

- Richer data model than key/value, but
- Rudimentary indexing options
- Limited query capabilities

<table>
<thead>
<tr>
<th>row keys</th>
<th>column family “contact”</th>
<th>column family “personal”</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>Will</td>
<td>“bio”: “Will attended …”</td>
</tr>
<tr>
<td></td>
<td>“twitter”: “wlaforest”</td>
<td>“picture”: …</td>
</tr>
<tr>
<td></td>
<td>“email”: “<a href="mailto:will@10gen.com">will@10gen.com</a>”</td>
<td>“phone”: “555-555-5555”</td>
</tr>
<tr>
<td>row</td>
<td>Chris</td>
<td>“bio”: “ … ”</td>
</tr>
<tr>
<td></td>
<td>“email”: “<a href="mailto:chris@10gen.com">chris@10gen.com</a>”</td>
<td>“picture”: …</td>
</tr>
<tr>
<td></td>
<td>“phone”: “555-555-5555”</td>
<td>“hobby”: “golf”</td>
</tr>
</tbody>
</table>
Document Oriented

- Data modeled as documents or objects (XML and JSON)
- Richest and most natural data model
- Typically more sophisticated indexing and querying

```json
{  
  ssn: "123-45-6789",
  name: "John Terry",
  position: "CB",
  doctors: [
    {  
      name: "Dr Seuss",
      address: "123 Diver St",
      phone: "555-555-5555"},
    {name: "Dr No"},
    medicine: ["Lipitor", "aspirin", "beer"]
  }
}
```

```json
{  
  ssn: "987-65-4321",
  name: "Ryan Giggs",
  weight: 158,
  position: "M",
  doctors: [
    {name: "Dr Awesome"}],
  therapy: "yoga"
}
```
MUMPS revisited

- The data model is very similar to document oriented
  - Imagine slicing the top level array into separate documents

- Collapsing of language and data had advantages
  - Remove impedance mismatch

- The language not well suited for modern tasks and development

- Horizontal scale not part of original design

- GT.M and Cache are helping address short comings
  - MUMPS database engine + Language bindings
  - Some replication and data partitioning
MongoDB Point Case
As a document store shares parallels with MUMPS

- Schema agnostic
- De-normalized data model
- Individual records thought of hierarchically
- Healthcare records map naturally
  - Typically already in a document format (XML)

What of the data to programming language mapping?

- Seeks similar benefits of a MUMPS style coupling without the drawbacks
Drivers

Drivers for most popular programming languages and frameworks

13 official
20+ community created

Shell

Command-line shell for interacting directly with database

```javascript
> db.collection.insert({
  company: "10gen",
  product: "MongoDB"
})

> db.collection.findOne()

{
  "_id": ObjectId("5106c1c2fc629bfe52792e86"),
  "company": "10gen",
  "product": "MongoDB"
}
```
Emphasis on Distributed Computing
Automatic Sharding

- Increase or decrease capacity as you go
- Automatic balancing
- Optimized for commodity servers and cloud infrastructure
High Availability

- Automated failover
- Add or remove nodes dynamically
Global Data Distribution
Read Global/Write Local
Operational and Analytical Workloads

- Application interacts with primaries
- BI tools read from secondaries
- Workloads are isolated from one another
- Working set appropriate for each application
The Future
Heterogeneous Environment

- Foreseeable future will have RDBMS, NoSQL and MUMPS
- VISTA platforms built on top of MUMPS
- Many other MUMPS systems
- Majority of apps in healthcare space still built on RDBMS
- Many net new apps will move to modern NoSQL offerings
  - MUMPS is great database, but
  - Industry adoption of NoSQL
  - Rate of technical advancement & development
  - Distributed cost
  - Workforce skillset
  - Ecosystem of tools
Backup Slides
References

- Indeed.com job trends (http://www.indeed.com/jobtrends)
- Universal NoSQL Engine (http://bit.ly/14MbZqN)
- MongoDB manual (http://docs.mongodb.org/manual/)
var p = {
  author: "roger",
  date: new Date(),
  text: "Spirited Away",
  tags: ["Tezuka", "Manga"],
  pages: 345
}

> db.posts.save(p)
Querying

>db.posts.find()
>db.posts.find({'author': "roger", 'tags': "Manga"})
>db.posts.find({'pages': {$gt: 300}})
>db.posts.find({'$or': [{'author': "roger"}, {'tags': "Manga"}]})

```
{
    _id: ObjectId("4c4ba5c0672c685e5e8aabf3"),
    author: "roger",
    date: "Sat Jul 24 2010 19:47:11 GMT-0700 (PDT)",
    text: "Spirited Away",
    pages: 345
    tags: ["Tezuka", "Manga"]
}
```
// Simple secondary index
> db.posts.ensureIndex({author: 1})
> db.posts.find({author: "roger"})

// Index nested documents
> db.posts.ensureIndex({"comments.author":1})
> db.posts.find({"comments.author":"Fred"})

// Index on tags
> db.posts.ensureIndex({tags: 1})
> db.posts.find({tags: "Manga"})

// geospatial index
> db.posts.ensureIndex({"author.location": "2d"})
> db.posts.find("author.location" : { $near : [22,42] })
Atomic Update Operations

- $set, $unset, $inc, $push, $pushAll, $pull, $pullAll, $bit

```javascript
> comment = { author: "fred",
            date: new Date(),
            text: "Best Movie Ever"}

> db.posts.update( { _id: "..." },
                  {$push: {comments: comment}} );
```
Nested Documents

```json
{
  _id: ObjectId("4c4ba5c0672c685e5e8a3bf3"),
  author: "roger",
  date: "Sat Jul 24 2010 19:47:11 GMT-0700 (PDT)",
  text: "Spirited Away",
  tags: ["Tezuka", "Manga"],
  pages: 345,
  comments: [
    {
      author: "Fred",
      date: "Sat Jul 24 2010 20:51:03 GMT-0700 (PDT)",
      text: "Best Movie Ever"
    }
  ]
}
```