Managing Your Database

Understanding Distributed Databases
Related to Chapter 18 from course text

Andy Pollino

Source: V. Gudivada et al., Renaissance in Database Management: Navigating the Landscape of Candidate Systems, IEEE Computer, 49(4), April 2016.
The Problem

- We have large databases
  - Facebook has billions of writes
- We have diverse applications
  - The Mom & Pop app shop doesn’t need strict ACID adherence
- We want to run in real time
  - Waiting six minutes to get Twitter updates defeats the purpose
- We do not want to pay for RDBMs
  - Open-source systems allow Yik-Yak to experiment with new ideas
The Solution

● NoSQL systems
  ○ Don’t adhere to relational model of RDBMSs
  ○ Sacrifice data integrity for speed
  ○ Perform quickly - in real time
  ○ Scale well
  ○ Big data, Web 2.0, mobile computing

● Open-source RDBMSs
  ○ MySQL with middleware
  ○ Middleware allows for flexibility and speed as necessary
A (Cloud-based) NoSQL Approach

- Data replication to ensure high availability
- Version support for mobile applications
- Real time querying for attention spans
- Multiple, ad hoc query methods for app versatility

These needs lead to the conclusion that NoSQL systems are not perfect right away, but they evolve alongside the application with updates so that each system fits a specific application. They often avoid strict ACID compliance.
**Database Management System Subclasses**

- Oracle, MySQL, Microsoft SQL Server: popular RDBMS
- Clustrix, VoltDB, TokuDB, Spanner: all NewSQL systems
- Key value: Redis, MemcachDB, Riak
- Column family: Cassandra, HBase
- Document model: MongoDB, Couchbase
- Graph model: Neo4J, OrientDB, Virtuoso
- Native XML: MarkLogic, eXist-db, BaseX

1. Reading supplemented with Wikipedia's NoSQL article for example systems: [https://en.wikipedia.org/wiki/NoSQL](https://en.wikipedia.org/wiki/NoSQL)
## System contrasts

<table>
<thead>
<tr>
<th></th>
<th>RDBMS</th>
<th>NoSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Denormalization</strong></td>
<td>Minimize redundancy</td>
<td>Eliminate join operations</td>
</tr>
<tr>
<td><strong>Schema Design</strong></td>
<td>Rule-adherent</td>
<td>Coupling with applications</td>
</tr>
<tr>
<td><strong>Integrity Constraints</strong></td>
<td></td>
<td>No specification of data-value constraints</td>
</tr>
<tr>
<td><strong>Design transparency</strong></td>
<td>Transparent in performance</td>
<td>Application must know query format</td>
</tr>
<tr>
<td><strong>Join Operations</strong></td>
<td></td>
<td>Application must work for many-to-many</td>
</tr>
<tr>
<td><strong>Consistency</strong></td>
<td>Rigid</td>
<td>Eventual</td>
</tr>
<tr>
<td><strong>Installation/maintenance</strong></td>
<td>User friendly</td>
<td>Inhouse experts</td>
</tr>
</tbody>
</table>
Which one is right?

- Survey in cited reading
- Consistency, availability, partition tolerance tradeoff
Looking Forward

- NoSQL may bring stability, robustness, transaction volume
  - Effort to appeal to RDBMS vendors
- Subclass lines may be blurred
  - Every system tries to do everything
- NoSQL must become secure
  - Currently expects to operate on secure servers
- NoSQL must become a service
  - Currently too technical