The Alliance vs. The Horde
A struggle for the future of data

Michael Franklin

UMass Amherst
March 24, 2010

Database Philosophy

God made integers, all else is the work of man.
(Leopold Kronecker, 19th Century Mathematician)

Codd made relations, all else is the work of man.
(Raghu Ramakrishnan, former ACM SIGMOD Chair)
The Politics of Databases

• [The Relational Model] provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation on the other.

(E.F. Codd, CACM 1970)

In Other Words…

Relational DataBase Management Systems were invented to let you use one set of data in multiple ways, including ways that are unforeseen at the time the database is built and the 1st applications are written.

(Curt Monash, analyst/blogger)

That is, think about the data independently of any particular program.
Relational DB: Definitions

Relational database: a set of relations, each having:

- **Schema**: specifies name of relation, plus name and type of each column
- **Instance**: a table, with rows and columns.

Employee(eid: string, name: string, location: string, age: integer, salary: float)
Project (pid: string, pname: string, location: string, budget: float)
Assignment(eid: string, pid: string, end: date)

Can think of a relation as a set of rows or tuples.

Queries, Query Plans, and Operators

System handles query plan generation & optimization; ensures correct execution.

Issues: view reconciliation, operator ordering, physical operator choice, memory management, access path (index) use, …
DBMS Software Stack

- Query Optimization and Execution
- Relational Operators
- Files and Access Methods
- Buffer Management
- Transaction Management
- Disk Space Management

The Emerging Debate

Vs.
What is NoSQL?

- An emerging “movement” around non-relational software for Big Data
  - A “Tea Party” for data management
- Roots are in the Google (mostly) and Amazon homegrown software stacks
- Vibrant Open Source community
  - plus some recent converts
- Currently defined as what it’s not
- But then so wasn’t the “horseless carriage”

Some NoSQL Components

<table>
<thead>
<tr>
<th>Analytics Interface (Pig, Hive, …)</th>
<th>Imperative Lang (RoR, Java, Scala, …)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Parallel Processing</td>
<td>(MapReduce/Hadoop)</td>
</tr>
<tr>
<td>Distributed Key/Value or Column Store (Cassandra, Hbase, Voldemort, …)</td>
<td></td>
</tr>
<tr>
<td>Scalable File System (GFS, HDFS, …)</td>
<td></td>
</tr>
</tbody>
</table>
The Debate in a Nutshell
(from CACM Jan 2010)

From the Relational side:

“Engineers should stand on the shoulders of those who went before, not their toes.”

but then, a section on “Learning from Each Other”

From, the NoSQL side:

“The conclusions about performance in the comparison paper were based on flawed assumptions about MapReduce and overstated the benefit of parallel database systems.”

Points of Contention

<table>
<thead>
<tr>
<th>Issue</th>
<th>Relational</th>
<th>NoSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>claim: Yes</td>
<td>claim: YES</td>
</tr>
<tr>
<td>Schema/Structure</td>
<td>Crucial</td>
<td>Evil</td>
</tr>
<tr>
<td>Declarative/Imperative</td>
<td>Data Independence</td>
<td>Real Code</td>
</tr>
<tr>
<td>Consistency</td>
<td>Crucial</td>
<td>Optional</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>Coarse Grained</td>
<td>Fine Grained</td>
</tr>
<tr>
<td>Coolness (a.k.a. “Google Envy”)</td>
<td>What’s Google?</td>
<td>Dude!</td>
</tr>
</tbody>
</table>

Note: there is also a cost/TCO debate as well
The Issue: Scalability

Scalability (continued)

- Often cited as the main reason for moving from DB technology to NoSQL
- Most of such moves, however, are from “MySQL”
- DB Position: there is no reason a parallel DBMS cannot scale to 1000’s of nodes
- NoSQL Position: a) Prove it; b) it will cost too much anyway
The Structure Spectrum

- **Structured (schema-first)**
  - Relational Database
  - Formatted Messages

- **Semi-Structured (schema-later)**
  - Documents
  - XML
  - Tagged Text/Media

- **Unstructured (schema-never)**
  - Plain Text
  - Media

---

Schema: Teaching a Pig to Sing?

- “Pig Latin” [Olston et al. SIGMOD 08]
  - Why have a schema?
    1) Transactional (referential?) Consistency
    2) Fast point look ups through indexes
    3) Curation for future (other) users
  - Flexible, optional, nested data model
  - Data remains in files (no admin)

- “Column Family” models of BigTable, Hbase, Cassandra, CouchDB, …

- “Schema on Read”? == Errors on Read?
Whither Schemas?

DB: Schemas are necessary for correctness, robustness, and evolvability.

NoSQL: a) Schemas keep me from getting my job done. b) messy data is reality.

Other Structure Issues

- NoSQL: a) Tables are unnatural, b) “joins” are evil, c) need to be able to “grep” my data.
- DB: a) Tables are a natural/neutral structure, b) data independence lets you precompute joins under the covers, c) this is a price of all the DBMS goodness you get.

This is an Old Debate – Object-oriented databases, XML DBs, Hierarchical, …
Fault Tolerance

• DBs: coarse-grained FT
  – Fewer, Better nodes, so failures are rare
  – Transactions allow you to kill a job and easily restart it
• NoSQL: Massive amounts of cheap HW, failures are the norm and massive data means long running jobs
  – So must be able to do mini-recoveries
  – This causes some overhead (file writes)
• Agreement – easy to support both models

Other Issues

• Consistency
  – Application-dependent and religious
• Declarative vs. Imperative
  – Issues well-known
  – Bordering on religious at this point
• Cost
  – “Making Free Software Affordable”
  – Need to look at TCO not just initial cost
• Coolness
  – Clearly in the eye of the beholder…
Is There a Middle Way?

Scale Independence

- As a site’s user base grows and workload volatility increases:
  - No changes to application required
  - Cost per user remains constant
  - Request latency SLA is unchanged
- Key techniques
  - Model-Driven Scale Up and Scale Down
  - Performance Insightful Query Language
  - Declarative Performance/Consistency Tradeoffs
**PIQL + SCADS**

- **“Active PIQL”**
  (don’t ask)
- **PIQL: Query Interface & Executor**
- **Flexible Consistency Management**
- **SCADS: Distributed Key Value Store**

**RAD Lab 5-year Mission**

*Enable 1 person to develop, deploy, operate a next-generation Internet application at scale*

- Multi-area faculty, postdocs, & students
  - Systems (Fox/Katz/Patterson)
  - Machine Learning (Jordan)
  - Networks (Shenker/Stoica)
  - Security (Joseph)
  - Databases (Franklin)
SCADS: Scale Independent Storage

• A self-managing distributed key value store
  – Simple API – put, get, ...
• Challenges in scaling stateful systems
• Model-based management
  – Based on Workload not Observed Latency
  – Replicate, Split, Coalesce
• Dynamic scaling up and down
• Supports Latency-based Service Level Objectives (SLOs)
Data storage configuration

- Shared-nothing storage cluster
  - (key,value) pairs in a namespace, e.g. (user,email)
  - Each node stores set of data ranges,
  - Data ranges can be split until some minimum size promised by PIQL, to ensure range queries don't touch more than one node

Workload-based policy

Policy input:
- Workload per histogram bin
- Cluster configuration

Policy output:
- Short actions (per bin)

Considerations:
- Performance model
- Overprovision buffer
- Limit actions to X kb/s
Cost-comparison to fixed and optimal

- Fixed allocation policy: 648 server units
- Optimal policy: 310 server units

<table>
<thead>
<tr>
<th>Overprovision factor</th>
<th>get/put SLA (ms)</th>
<th># server units</th>
<th>% savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>180/250</td>
<td>358</td>
<td>48</td>
</tr>
<tr>
<td>0.6</td>
<td>140/225</td>
<td>389</td>
<td>40</td>
</tr>
<tr>
<td>0.7</td>
<td>120/200</td>
<td>422</td>
<td>35</td>
</tr>
</tbody>
</table>

PIQL: a Performance Insightful Query Language
Is there a Middle Ground?

- **SQL**
  - expressive, powerful + logical and physical data independence
  - But, **Performance Opaque**
    - easy to write expensive queries
    - easy to write queries that become more expensive over time
  - Also, data “locked away” (can’t grep it)

- **NoSQL** – imperative, schema-later, distribute-it-yourself
  - cost of operations is explicit

---

**PIQL**

- Scale-Independent language subset
  - Only allow queries where we can calculate a bound for the number of underlying operations it will perform
  - Predeclare query templates: Optimizer decides what indexes are needed (i.e., materialized views)

- Performance feedback given by compiler
  - Unbounded queries are disallowed
  - Queries above specified threshold generate a warning

- Provides: **Bounded** number of operations

- + Strong SLOs = Predictable performance?
DDL

ENTITY User {
  string username,
  string password,
  PRIMARY KEY(username)
}

ENTITY Subscription {
  boolean approved,
  string owner,
  string target,
  FOREIGN KEY owner REF User,
  FOREIGN KEY target REF User
  MAX 5000,
  PRIMARY KEY(owner, target)
}

ENTITY Thought {
  int timestamp,
  string owner,
  string text,
  FOREIGN KEY owner REF User
  REFERENCES User
  PRIMARY KEY(owner, timestamp)
}

Queries

PIQL OVERVIEW: QUERIES

- Must be declared ahead of time
- Can be parameterized like stored procedures
- Allows SQL-like WHERE, ORDER BY and LIMIT

QUERY findUser
  FETCH User
  WHERE name = {1:name}

- Joins can be performed as long as intermediate/final results are bounded

QUERY subscribers
  FETCH user
  OF subscription BY target
  OF user me BY owner
  WHERE me={this}

Called on a “User” object, e.g. “person.subscribers”
More Queries

**PIQL OVERVIEW: QUERIES**

- Even more complicated queries like Twitter's `timeline` can be expressed

```
QUERY thoughtstream
FETCH Thought
    OF User | BY owner
    OF Subscription | BY target
    OF User | me | BY owner
WHERE me=[this] AND approved = true
ORDER BY timestamp
LIMIT [1:count] MAX 100
```

"Return the most recent thoughts from all of my "approved" subscriptions."

Operations are bounded via schema and limit max

---

**PIQL/SCADS Scalability**

![Graph showing query latency and fraction of queries for different machine configurations (20, 50, 100 machines).]
PIQL + SCADS Wrap Up

• Goals are “Scale Independence” and “Performance Insightfulness”
• SCADS provides scalable foundation with SLA adherence
• PIQL uses language restrictions, schema limits, and precomputed views to bound # of SCADS operations per query.
• These work together to bridge the gap between “SQL” and “NoSQL” worlds.

Reprise: Alliance vs. Horde

The Horde aren't evil for just being the Horde, neither are the Alliance all pure as newly fallen snow. Each faction has done its fair share of pure and evil acts depending on the actions of individual heroes and the motivations of their leaders. [Link](http://www.wow.com/2009/04/26/the-state-of-the-horde-and-the-alliance)

The Alliance: Humans, Dwarves, Night Elves, Gnomes

The Horde: Blood Elves, Orcs, Trolls, Undead
Concluding Thoughts

• The Relational Model has been surprisingly controversial for over 40 years!
• There are new, real problems that the existing DB stack does not address well
• A huge spike in data management creativity both outside and inside the database world
• Initial engagements have been rough but sides are starting to talk
• A growing realization that NoSQL is probably NOSQL (i.e., “Not Only SQL”)