

NoSQL – December 2009

Charlotte JUG

Tyler Williams
Principal Consultant @ DataTerrace

My Background

- Independent Consultant for 15+ years
- 8 years Java
- 10 years Oracle
- Frameworks
- Database Integration

Credits/Links

Several slides were leveraged with permission from Emil Eifrem of Neo4j.org

<http://www.slideshare.net/emileifrem/nosql-east-a-nosql-overview-and-the-benefits-of-graph-databases>

Product References

Neo4J

Cassandra

Hadoop

Pig

Cascading

CouchDB

MongoDB

HBase

Voldemort

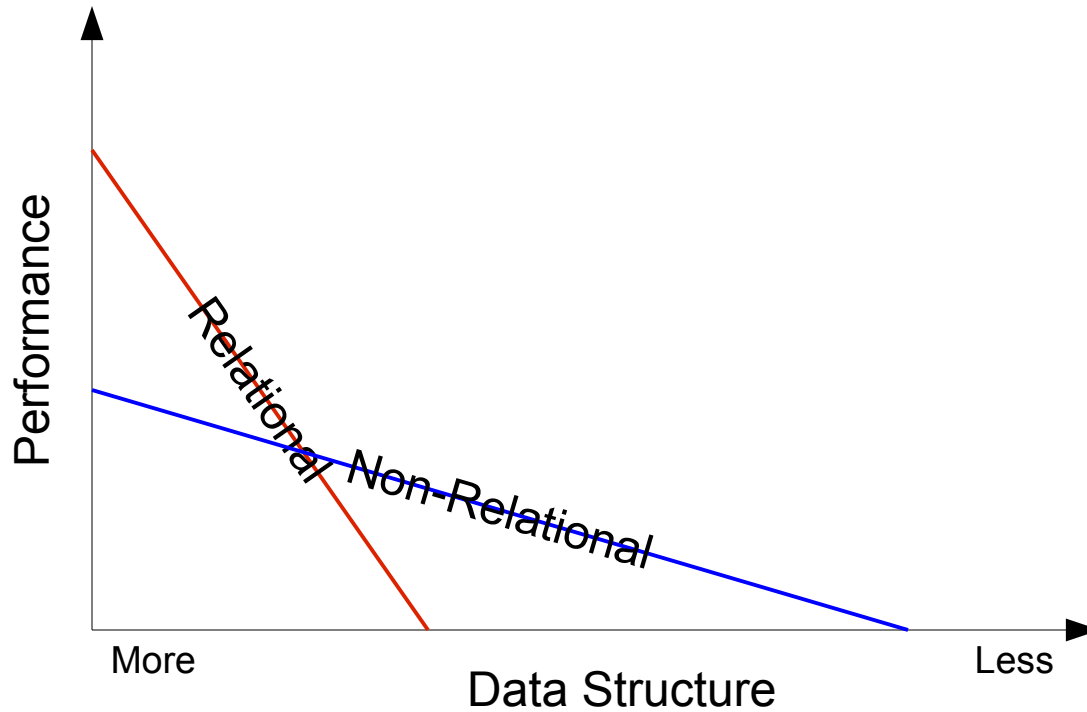
Riak

VertexDB

NoSQL

- Not Only SQL
- Why now?
 - Data set size
 - Semi-structured nature

NoSQL



Pros/Cons

- + No O/R mismatch (Document & Graph)
- + Easy to Evolve/Migrate Schemas
- + Semi-structured Data Representation
- Nascent Tool Support
- Vendor Lock-in
- Less flexible querying (if supported)
- Not fully (ACID) transactional (except Graph)

CAP Theorem

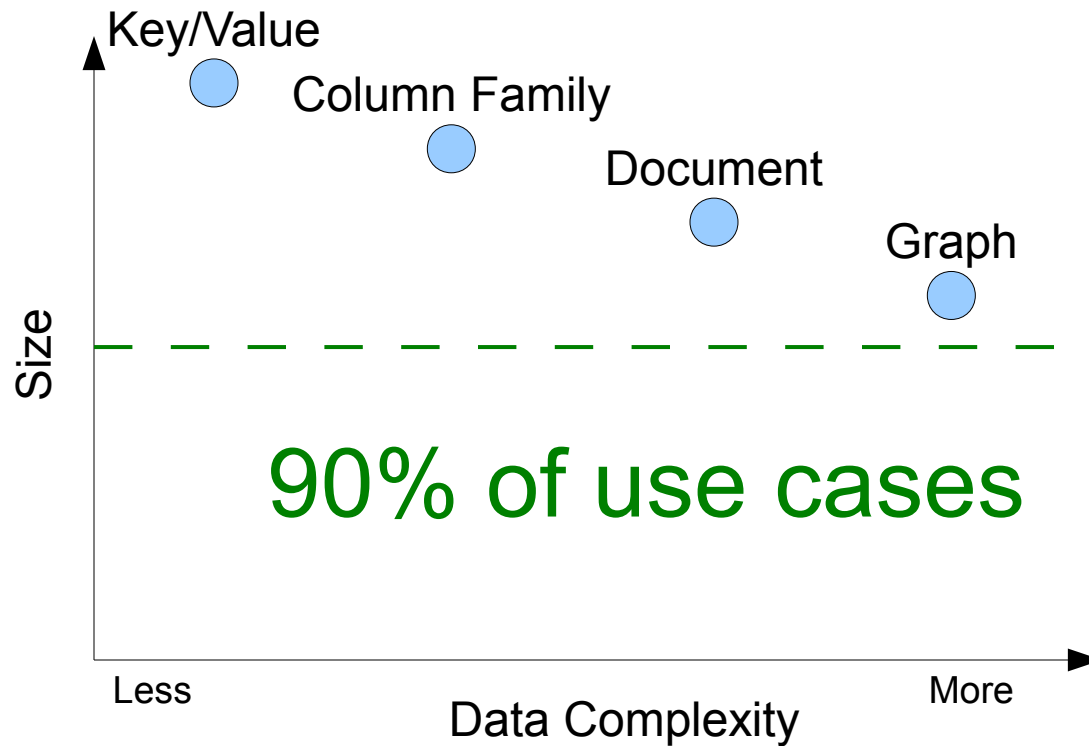
Pick 2 of 3

- Consistency
 - Strong – ACID
 - Weak – Eventually Consistent
- Availability
 - Node Failure Tolerant
- Network Partition Tolerance
 - Effect of lost Partition Connectivity

Categories

- 1) Key/Value
- 2) Column (Bigtable)
- 3) Document
- 4) Graph

Categories



Key/Value

The sweet spot for K/V stores is high performance and availability

- Voldemort
 - Support for complex compound objects
 - JSON
 - Replication & Partitioning
- Riak – K/V store & Document DB
 - Map/reduce engine
 - HTTP/JSON query interface
 - Tunable CAP
 - Write availability

Column Oriented

Semi-structured data

Sparse, distributed, persistent
multidimensional sorted map

- Cassandra
 - Highly Available, Decentralized, Fault Tolerant, Eventually Consistent
- Hbase
 - Inexpensive, Simple, Scalable
 - Massively large sets of data

Document

Typically defined by their use of JSON for storage and a queryable api. Nestable hashmaps.

- CouchDB
 - JSON
 - REST Interface
- MongoDB
 - BSON
 - Native APIs

Graph

Data is stored the same way it is used...Nodes and relationships

- Neo4J
 - Commercial
 - Transactional
 - Queryable
- VertexDB
 - Berkley License
 - Limited Ongoing Development Effort

Tips

- Immaturity
 - Security
 - Handling of Dates/Native Types
 - Tooling
- Longevity
 - How Many BigTable clones will survive
 - What does the community/commercial support
- Enterprise Capabilities
 - Workflow Support
 - Roll-your-own Rollback
 - Manual Data Mitigation for Eventual Consistency

Use Case

Flexibility (and Performance)

- Employee
- Jobs
- Salaries

Employee

```
String firstName  
String lastName  
List<Job> jobs
```

Job

```
String title  
Double salary
```

Use Case

Flexibility (and Performance)

- Employee
- Jobs
- Salaries

EMP		
ID	LNAME	FNAME
1	Jim	Smith

EMP_JOB		
EMP_ID	JOB_ID	SALARY
1	1	100
1	2	100000

JOB	
ID	TITLE
1	Peon
2	King

Use Case

Flexibility (and Performance)

```
select e.lname, e.fname, j.title, ej.salary
from EMP e, EMP_JOB ej, JOB j
where e.id = ej.emp_id
and ej.job_id = j.id
```

LNAME	FNAME	TITLE	SALARY
Smith	Jim	Peon	100
Smith	Jim	King	100000

Use Case

Flexibility (and Performance)

```
db.get('lname':'Smith')
```

```
{"_id" : "49903677516250c1008d624e" ,  
"lastName" : "Smith" , "firstName" : "Jim",  
"jobs":  [{ "Title": "Peon" , "Salary": 100} ,  
           { "Title": "King" , "Salary": 100000} ]}
```

Use Case

Flexibility (and Performance)

- Employee
- Jobs
- Salaries
- Tasks

EMP		
ID	LNAME	FNAME
1	Jim	Smith

JOB	
ID	TITLE
1	Peon
2	King

EMP_JOB		
EMP_ID	JOB_ID	SALARY
1	1	100
1	2	100000

TASK		
ID	JOB_ID	TITLE
1	1	Sweep
2	1	Cook
3	2	Rule

Use Case

Flexibility (and Performance)

```
select e.lname, e.fname, j.title, ej.salary, t.title as  
task_name
```

```
from EMP e, EMP_JOB ej, JOB j, TASK t
```

```
where e.id = ej.emp_id
```

```
and ej.job_id = j.id
```

```
and j.id = t.job_id
```

LNAME	FNAME	TITLE	SALARY	TASK_NAME
Smith	Jim	Peon	100	Sweep
Smith	Jim	Peon	100	Cook
Smith	Jim	King	100000	Rule

Use Case

Flexibility (and Performance)

```
db.get('lname':'Smith')
```

```
{ "_id" : "49903677516250c1008d624e" , "lastName" :  
  "Smith" , "firstName" : "Jim",  
  "jobs":  [ {"title":"Peon", "salary":100,  
             "tasks":[{"title":"Sweep"},  
                    {"title":"Cook"}],  
            {"title":"King", "salary":100000,  
              "tasks":[{"title":"Rule"}]}
```

Map Reduce

Analyzing large data sets using parallelization

- Apache Pig
 - Built on Hadoop
 - Pig Latin
- Cascading
 - Built on Hadoop
 - Java API

Cascading API

- **Pipe** - series of processing steps (parsing, looping, filtering, etc)
- **Flow** -the association of a pipe (or set of pipes) with a data-source and data-sink. In other words, a flow is a pipe with data flowing through it
- **Cascade** - the chaining, branching and grouping of multiple flows