
NoSQL
Jeff Conrad
07/08/2010
Images: Digital Blasphemy

NoSQL: DEFINITION

- Non-relational
- Distributed
- Open-source
- Horizontally scalable

- Not-only SQL refers to:
  - Class of non-relational storage
  - Do not require fixed schema, nor joins
  - Relaxes ACID properties

Why NoSQL?

> Web-scale databases (LinkedIn, Facebook, Google)
> Handle spikes (social, cloud)
> Throughput
> Limitation is Disk not CPU
> Shift from deterministic to probabilistic
> Frequent schema changes
> Open source communities

3 Papers to Read up on...

(2006 - Google)

2 Bigtable: A Distributed Storage System for Structured Data

http://labs.google.com/papers/bigtable-osdi06.pdf

(2000 - Brewer)

1 Towards Robust Distributed Systems

http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

(2007 - Amazon)

3 Dynamo: Amazon's Highly Available Key-value Store

http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

ACID vs. BASE Acronyms

- Atomicity
- *Consistency*
- *Isolation*
- Durability

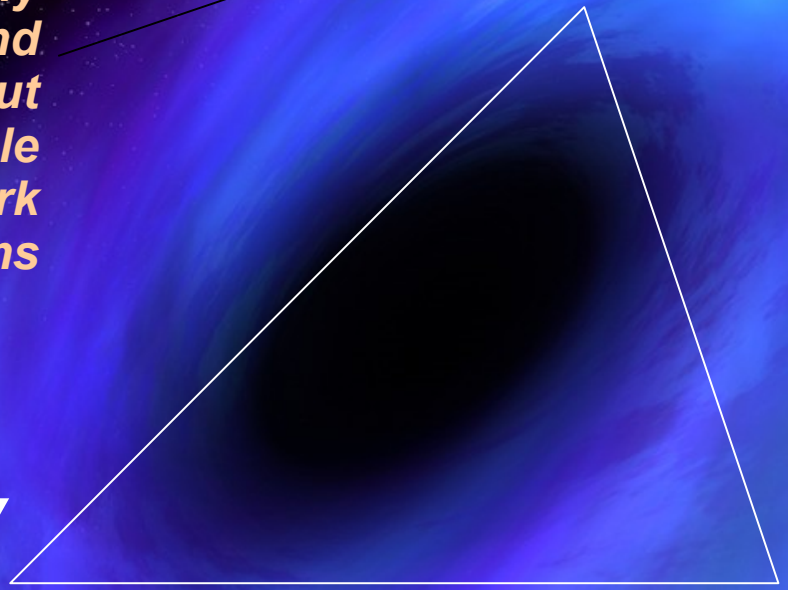
- Basically
- Available
- Soft-state
- Eventual Consistency

ACID vs. BASE EFFECTS

- > Consistency
- > Isolation
- > Focus on commit
- > Nested transactions
- > Availability?
- > Conservative
- > Pessimistic
- > Schema
- > Slow Evolution

- > Stale data OK
- > Availability first
- > Best Effort
- > Approximate answers OK
- > Aggressive
- > Optimistic
- > Simpler
- > Faster evolution

The **CAP** Theorem



Consistency

Availability

Partition tolerance

BigTable is a CA system; it is strongly consistent and highly available, but can be unavailable under network partitions

Distributed databases with pessimistic locking

Dynamo is an AP system; it is highly available, even under network partitions, but eventually consistent.

"You can have at most two of these properties for any shared-data system"
- Brewer

Brewer's Conclusions

- > Availability
- > Evolution
- > Graceful Degradation
- > Think probabilistically
  - > Working < 100%
  - > Fault tolerance < 100%
  - > Partial results OK, better than none
- > Capacity x completeness = constant

NoSQL: Wide Column Store

- > Hadoop / HBase
- > Cassandra (Facebook)
- > Hypertable
- > Cloudera

NoSQL: Document Store

- > CouchDB
- > MongoDB
- > Riak
- > Terrastore
- > ThruDB
- > OrientDB
- > RavenDB

Lists: nosql-database.org

NoSQL: Key Value / Tuple Store

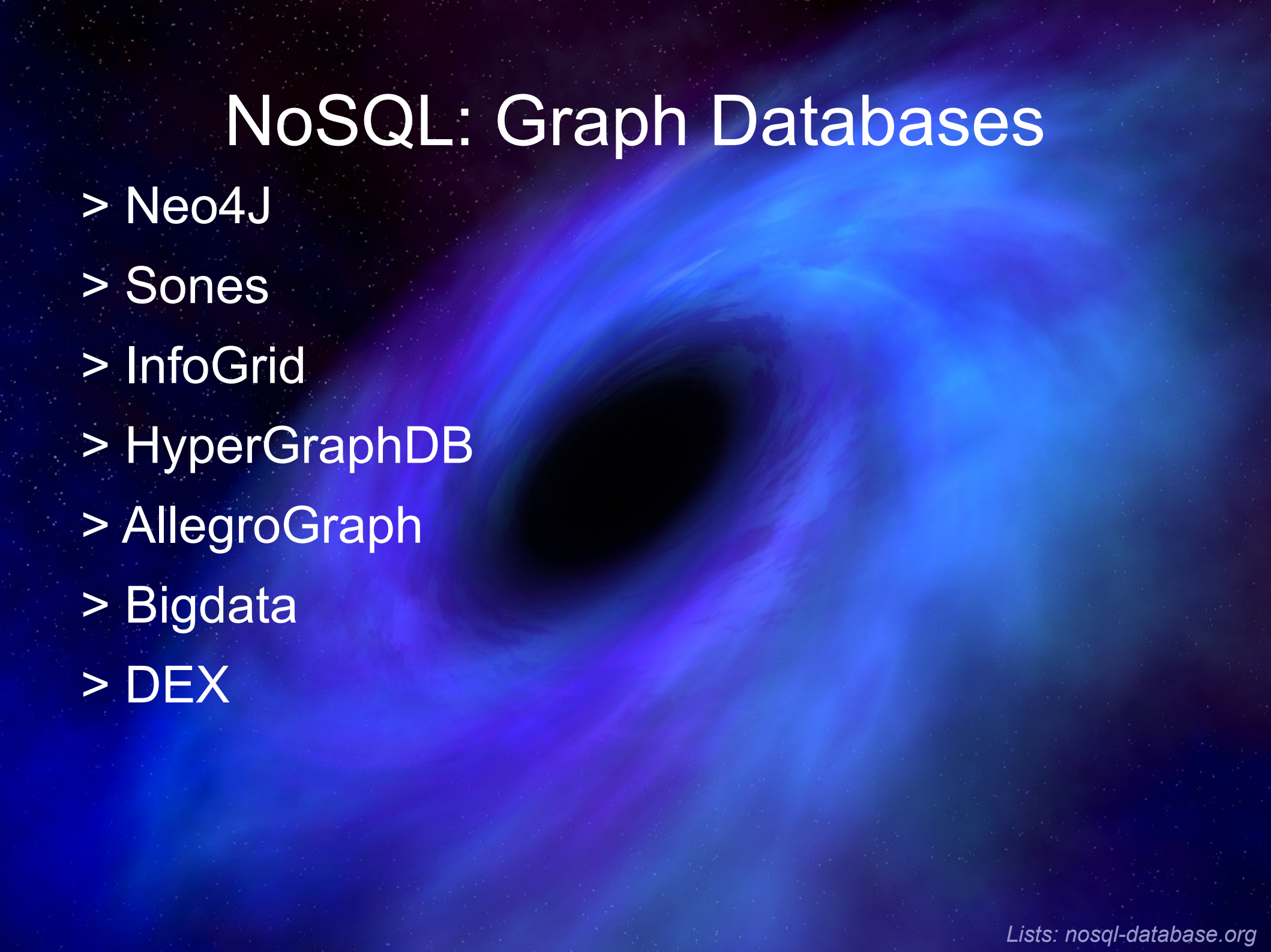
- > Amazon Simple DB
- > Azure Table Store
- > Chordless
- > Redis
- > Scalaris
- > G. T. M
- > Scalien
- > Berkeley DB
- > MemcacheDB
- > HamsterDB
- > Pincaster
- > GenieDB

Lists: nosql-database.org


NoSQL: Key Value / Tuple Store
> Amazon Dynamo
> Voldemort
> Dynomite
> KAI
Lists: nosql-database.org

NoSQL: Graph Databases

> Neo4J
> Sones
> InfoGrid
> HyperGraphDB
> AllegroGraph
> Bigdata
> DEX



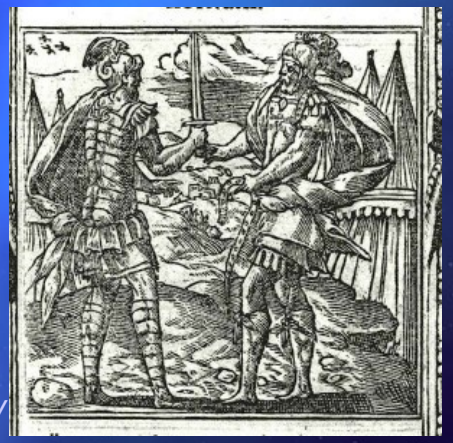
Lists: nosql-database.org

Apache Cassandra/Hector/Thrift

> Cassandra is a distributed database based on Dynamo and Bigtable

> Hector is a Java Cassandra Client

> Thrift is a systems interface to open services to multiple languages



Weaver, E. Up and Running with Cassandra.
http://blog.evanweaver.com/articles/2009/07/06/up-and-running-with-cassandra/

Weaver, E., Cassandra data model misconceptions, and their sources.
http://www.mail-archive.com/cassandra-dev@incubator.apache.org/msg00732.html

WTF is a SuperColumn? An Intro to the Cassandra Data Model by Arin Sarkissian
http://arin.me/blog/wtf-is-a-supercolumn-cassandra-data-model

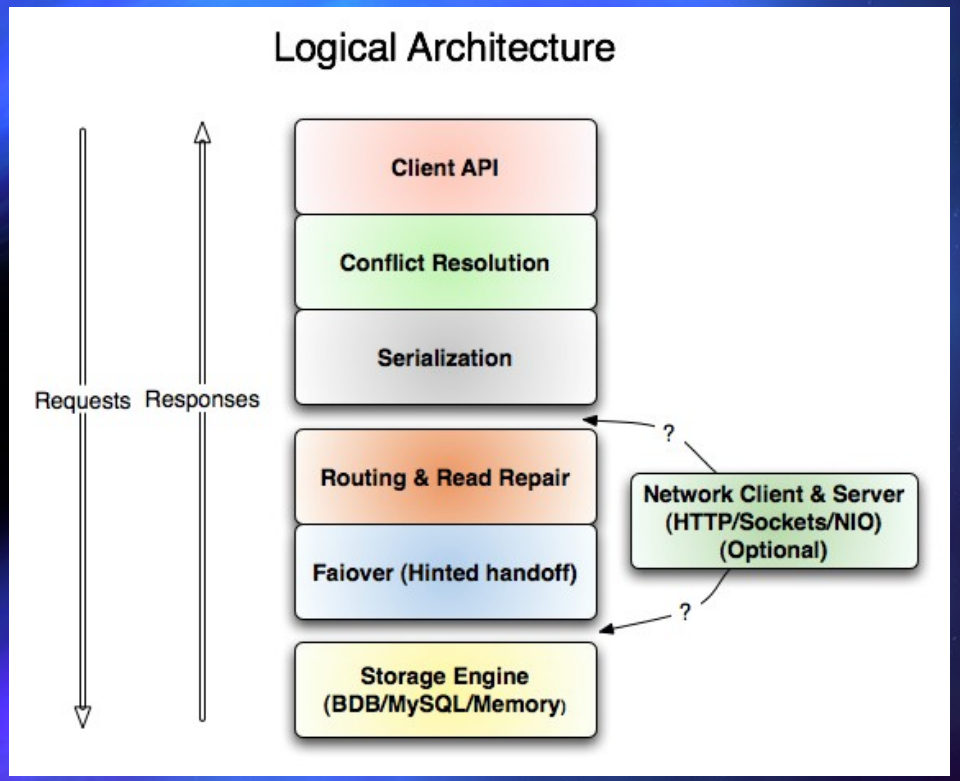
Lists: nosql-database.org

Cassandra Terminology

- > Column = key-value pair + timestamp (attribute)
- > Super Column = map of attributes (row)
- > Standard Column Family = map of rows (table)
- > Super Column Family = map of tables (table of tables)
- > Keyspace = map of column families (database)

Voldemort

- no complex query filters
- all joins must be done in code
- no foreign key constraints
- no triggers



Voldemort: Cons and Pros

> Only efficient queries are possible, very predictable performance

> Easy to distribute across a cluster

> Service-orientation often disallows foreign key constraints and forces joins to be done in code anyway (because key refers to data maintained by another service)

> Using a relational db you need a caching layer to scale reads, the caching layer typically forces you into key-value storage anyway

> Often end up with xml or other denormalized blobs for performance anyway

> Clean separation of storage and logic (SQL encourages mixing buisiness logic with storage operations for efficiency)

> No object-relational miss-match

> No complex query filters

> All joins must be done in code

> No foreign key constraints

> No triggers

Don't Forget

> Backups and Recovery
> Capacity Planning
> Performance Monitoring
> Data Integration
> Tuning and Optimization