## Overview

No normalization

Somewhat of a return to file-based databases

Trade off’s is a custom theme

Most implementations are open source

Open source dev’s tend to like using NoSQL more than those used to using proprietary / relational db’s

Most are distributed

Setting up is not as complicated as in a relational system

Replication

Load balancing

Fault tolerance

Key traits

Not relational

Schema free / flexible

Eventual consistency

Processing a request and implementing a request are two different things

Much processing is asynchronous

Many engines delay actually doing the work

With clustering, changes you’ve submitted may take awhile to actually propogate

Web scale

NoSQL is often used as backend to huge customer facing web apps

Yahoo / Google / Facebook / LinkedIn

Their sites typically have to implement very special proprietary extra steps to use NoSQL with the volume they have to support.

NoSQL db’s main purpose is to remove the structure and constraints imposed by structured fixed schema

Useful where database needs aren’t too complex, just need to perform mathematical / analytical operations on large amounts of data very quickly.

Common traits

## Consistency

Important since they are typically distributed

NoSQL values availability over consistency

Offers eventual consistency (like DNS propogation)

NoSQL may buffer requests

Relational db’s guarantee consistency at the expense of having to wait for the operation to return.

NoSQL returns control immediately.

TODO: Is “eventual consistency” guaranteed?

## Indexing

Most NoSQL db’s are only indexed by key of table

Secondary indexes (indexes by something other than key) are only supported by some NoSQL db’s.

The fact that schema’s are flexible can make the idea of secondary indexes seem strange...how can you index on something that might not even be there?

Often, primary key is physically clustered, meaning the

## Querying

Typically, NoSQL db’s don’t support a query language

Query languages typically allow clients a declarative access to all the data in the db

With NoSQL, you need to think of the db as a bunch of files.

The usual process is to have more logic in your program to iterate over files, process datasets, and generate the result.

Most NoSQL db’s use imperative code as opposed to declarative syntaxes.

## MapReduce

Used extensively with NoSQL DB’s

Code that contains two aspects, a map process and a reduce process.

Split up the work of a query.

Dispatch individual parts of query to individual nodes of cluster to be performed concurrently.

Results of individual parts of query get organized according to the key of the result.

At each location, the results get merged (reduce step).

Map step splits query up, reduce step takes individual results and merges them together into a smaller resultset.

Hadoop

Open source version of Google’s Map / Reduce proprietary solution

HDFS

Map Reduce engine

Wide column stores is a class of DB

HBase

Open source version of Big Table

HBase and Hadoop are often used together

Lots of map reduce code executed by Hadoop gets executed on HBase tables

Amazon Elastic Map Reduce

Cloud provisioned cluster running Hadoop map reduce

Can read / write from a variety of different datasources

Dynamo DB

S3

Relational DB’s

Map reduce is a processing mechanism

NoSQL files are valid inputs to Map Reduce processing

Hive

Add-on to Hadoop

SQL-like abstraction on top of Hadoop

Hive has its own HDFS structure

Can also work on top of an HBase table structure

## Sharding

Partitioning mechanism used in NoSQL

Related to the distributed nature of NoSQL

Each server in a cluster has separate partitions

Database is split across servers

Queries get fanned out across nodes using map reduce as the processing mechanism

Shards can be duplicated, so supports DR

Shards can be distributed so that certain shards are close to certain customers, almost like a CDR.

Smart distribution of shards allow for optimization of map reduce processing

## Four main subcategories

Key - Value stores

Most common, general purpose, and foundational

Less exotic than others

Suited to a wide variety of apps

Has rows

Schemas can vary across rows

The best example of schema freedom

Very used in the cloud

Amazon Simple DB

Table store on Windows Azure

Memcache DB, Voldemort

Memcache DB is different from MemcacheD

Dynamo DB (AWS)

Amazon web-services offering

In wide use at Amazon

Dynomite

Precedes DynamoDB, open-source

Rows have an id, then a bunch of key-value pairs

Rows can reference rows in separate tables, but it’s an unstructured mechanism (no fk’s)

Manual mechanics abound

Wide column stores

Have tables and columns in the tables

Diff from key/value in that they take a hybrid approach to defining table structures

Mix declarative schema of relational db’s with variable schema of key/value stores

Consistent schema

column families

Each column family has columns which are a K/V pair that can vary from row-to-row

Big Table (Google)

Proprietary at google

Used with it’s Hadoop computations

HBase (Hadoop)

Cassandra (Facebook)

column families (super columns)

tables (super column families)

Can be used with Hadoop

All these db’s have Hadoop components to their engines

HBase + Hadoop is the most capable Big Data capable solution

They’re all good for Big Data

Amazon’s Dynamo is an exception in that it is a K/V db and is capable of being used for big data applications

Tables have rows

Rows have rowid and super columns

super cols have columns

Every column must be part of a super column

Different rows can choose to not use all columns in a super column

Document stores

Have databases instead of tables

Have documents instead of rows

Documents are based on JSON

Each doc has properties with values

Values can be scalars, arrays, links to other docs

Documents can reference docs

Attachments can be added to documents

Old versions are retained

Good for content management

Some people think they’re just specialized key-value stores

Most popular with developers, startups

Very closely aligned with web-servers

CouchDB

MongoDB

Documents can contain documents

Documents link to other documents using real codified links

Docs can be addressed using URI

CouchDB has a complete REST interface

Docs are JSON objects

CouchDB / MongoDB use Javascript as the native language

CouchDB supports creation of view functions registered at their own URI, returning HTML

You could create and entire app in the database

Each doc has an id and a revision

Values of doc properties can be scalars, arrays, other documents

Docs have no intrinsic schema that can be used to enforce structure...you need to enforce everything manually.

Documents can be assigned special id’s, like \_design/foo to support access via REST api

A document with id \_design/foo having views property having value of JSON having key “all” and value JSON obj having key “map” and value function(doc) can be accessed at URI http://host:port/dbname/\_design/foo/\_view/all

Generates JSON output.

The same doc may have shows property having JSON value having key summary and value function(doc, req)

http://host:port/dbname/\_design/foo/\_show/{docid}

Generates HTML

Graph database

Useful for expressing relationships between data

Nodes

Rows in a table

Can have properties and values

Edges

Join nodes together

Like

Neo4j

Database

Object

Address relationship with Address object

Friendship relatioship with Jane

Purchase relationship with Purchase X

NodeSQL as a killer app?

Which subcategory is best for different scenarios?

Making a case for each subcategory

Content management

Doc db’s work well here

Content

has meta-data

K/V great for meta-data

Attachments are great for non ascii, scalar data

Versioning and URI addressibiity is great

Easy to rollback with versioning

Couch DB is a web db, great for web apps

Web apps meaning apps that are used on internet, not an enterprise corporate system

Product Catalog

Metadata

Things have many similar attributes,

Then there are class-specific attributes only present in certain rows

K/V stores and Wide column stores work well

K/V better if schema changes often since there’s no fixed schema declared

Wide column stores require that at least column families be declared

Social Networking apps

Great for graph databases

Objects and relationships

Friendships, threaded interactions, conversations, comments

Membership and Ownership

Avoids self-join and many to many joins in relational databases

Relationships are modeled much nicer than in relational db’s

Big Data apps

Wide-column and K/V stores work well

Map Reduce is ideal for this.

Chopping up problem into smaller things

Compute individual problems

Hadoop and HBase are the ideal

Sharding is very helpful

Allows scaling

Allows HA

Anything requiring Analytics and BI

Analytics and BI are read-only operations

No need for structured queries since you’ll just want to do huge table scans and NoSQL lets you do that real quickly

Read-only stuff

Misc

Anything that produces a lot of data

Event-driven logs

User-profiles / preferences

ini files

Configuration

Mail, Status message streams

Twitter timelines

Web data

Automobile directions

Info for locations in a map

User reviews

## Where do relational DB’s fit in?

Relational DB’s aren’t obselete

They’re indispensable in many instances

Scenarios where relational DB’s are good fits

Transactions

Business systems require atomic transactions

Multiple individual changes can be applied and rolled back atomically

You can’t process an order without decrementing inventory

Accounting

Can’t apply a credit without also applying the debit atomically

Almost always isn’t good enough

Formal schema

Regular business processes rely on formal schema

Business processes thrive on structure

Structure prevents errors

Stocks / trades

PO line items

purchase orders must be standardized

Personnel records

Insurance policies

Anything that originates from paper forms

Anything that doesn’t really need Map Reduce, document or graph representation

Operational systems

After time of capture, we can load data into a NoSQL database and perform analytics.

At time of capture, there’s no need for this.

UI’s expect consistent binding to data fields

Can’t declare a screen layout without having a fixed schema

Strong typing (beans) requires a fixed schema

ORM

Declarative queries

Relational DB’s are critical for running ad-hoc queries.

It’s unreasonable to expect a separate procedural program be written for each possible ad-hoc query. Impractible.

Relational db’s are great at optimizing access paths for ad-hoc queries

Database engine takes advantage of fixed schema

Imperative queries are fine if there’s only a few known queries to be performed

If queries are limited to the UI, it can work

If range of questions about data is unlimited, relational db becomes more useful

The fact that relational db’s use stored proc’s doesn’t mean that relational db’s are used imperatively. Behind the scenes, the queries are still written in a declarative manner.

Banded reporting

Banded report - one in which alternate rows have different colors

Detail sections and summary sections

Creating report with complete data is hard if you don’t know what the schema is

Reports are rooted in paper forms, and structured materials

Reports can be written, but they have to be generic based on the lack of solic schema requirements

Operational business processes almost always require relational databases.

## Dealing with lack of joins

How do you relate data entities to each other?

You need to store both pieces of data together

Requires denormalizing data

Data models in NoSQL seem to more resemble a data warehouse than anything else.

Data relationships are materialized in a number of different tables depending on anticipated usage

How do NOSQL db’s deal with updating data when that data is stored in multiple places?

TODO

## NoSQL, Relational, or Both

Criteria for choosing

Type of app your building

Choice comes down to whether consistency in DB is more important or scale is more important.

Another factor is whether this is internal vs external

external db’s often lead to nosql

Application for the data, or data for the system

If your site is about food

If your site is a backend for a food industry organization, then your maintaining a database and building an app to help facilitate it. => relational

Below a certain threshold of concurrent usage, NoSQL may be even slower than RDBMS.

Productivity

NoSQL db tooling is immature

NoSQL is much newer than RDBMS

In a large corporation, where speed of performing menial operations is key, NoSQL might be too unproductive

Queries for NoSQL db’s require a lot of work, including map/reduce work, programs, etc.

Where the app is the centerpiece as opposed to the data, you’ll likely only have a few queries, and the work required to implement the queries will make sense

Programming platforms, frameworks and components all favor RDBMS

Frequent schema changes may support a NoSQL solution

Skill sets and investment

If employees already have RDBMS skills. If you have a ton of RDBMS skills, it might be painful to move away from investing in that asset.

If you have tons of RDBMS infra, and licenses

If you have lots of apps written in RDBMS

How willing are you to replatform?

If your devs pref NoSQL, are a startup, are younger staff

cultural preferences are important

think about cost of morale

Does availability / scalability make RDBMS investment quesitons moot?

databases

Good for tracking relationships between entities

Key-value is used throughout all subcategores.

## Concepts

HBase

Uses HDFS

All writes are logged and buffered

Logged writes are batched

Every once in awhile, batches will be processed, and the physical files will be completely regenerated and ordered from scratch.

Control comes back quickly, changes will get applied later.

HDFS (Hadoop Distributed File System)

An append-only file system

No random access is allowed to files

## Terms

CAP Theorem

db’s may only excel at two of following three attributes: consistency, availability, partition-tolerance

relational db’s tend to excel at consistency and partition-tolerance

NoSQL db’s tend to excel at availability and partition tolerance

ACID

NoSQL doesn’t support ACID

Atomicity, consistency, isolation, durability

Relational db’s can guarantee

Clustering index

database index which reorders the way that data is physically stored