2013/10/28

Managed designs – Mauro servienti

Mauro.servienti@Manageddesigns.it

Goals:

What is raven

How to use it

How to avoid screwing up

Comparison with other technologies

Problem: orm misuse

Polyglot persistence:

* several persistence technologies for different kinds of data
* need for it is a function of how many use cases a product implements (e.g. erp)

Example: Analysis services + oltp

* Uses ad-hoc data structure fit for a very specific purpose

Driver behind sql/e-r model adoption: hardware cost

Storage cost estimation should include backup costs

Examples of document-oriented db:

* Exchange
* Active directory
* Lotus notes

RavenDB exploits a **native windows** component: name?

Relational model problem: scaling (two examples)

* Performance (essential for cloud/saas)
* High availability (e.g. different timezones, slecht network prestaties) (essential for cloud/saas)

CAP Theorem

Consitency high performance high availability, you can only achieve TWO

(e.g. e-r model use case: high read/writes on same table)

Possible solution: eventual consistency (but how fast????)

e.g. you have inserted an invoice even though you can’t see it ☺ (i.e. ui-related problem)

PROBLEM: change orm isolation level depending on query, orm does not encourage you to think about it

**Missing in RDBMS:**

Failure mode (distributed data, one machine fails, what then?)

Write on a, read from b , data must be the same because we want consistency but what about performance???

**Sharding**: (feature-based data partitioning technique)

Example: geographically distributed data (2 branches: London, Milan, each one stores their orders in a geographically close location, distributed queries are transparent to the application layer)

Advantages: paged distributed queries

Application configures query behavior in case of failure e.g. a remote system is down:

* I want an exception
* I want partial results

Failed rdbms implementations

* Xml columns: kill rdbms
* Azure Table storage (limited by underlying SQL server)

(odata schema)

Document-based db are NOT hot-swappable

(e.g. nhibernate + user chooses backend)

Persistency-ignorance is not easily achieved, true for reading modules

Attributes: eav/vertical table

SQL Server solution: cte

Problem: how to get Nhibernate to understand it? 🡺 user type + dialect extension

Document-based db: you get it for nothing

**NOSQL**

Denormalised data, downside: storage needs, synchronisation

No schema, advantages: no big up front design + painless to change data models

Data versioning:

* Rdbms: application version tightly coupled to db schema (must deploy at the same time)
  + Case study: multinational clothes retailer
  + How can I update my backend???? I am screwed! (downtime 🡺 lost sales)
* Document-based db: incremental schema update + application is responsible for saving data in the correct format (i.e. read version 1.0 save version 1.1, use metadata to track version information, avoid polluting domain-related information with infrastructure-related information (i.e. versioning) )
  + Problem: your application manages data versioning
  + **Golden rule: no two applications use same db (enforce separation of application lifecycles, a soa tenet)**
    - **How can two applications share data safely? 🡪 apis**

Many if not all nosql dbs use json as data persistence format

Problem: json typesystem is limited, two different applications may interpret same data differently.

Sacrifice consistence to performance and scalability.

Ravendb: Master goes down, his slave becomes readonly

Cassandra/redis/couchdb: key/value pairs, combine with elastic search to perform lookups

Graphdb: look at relationships between different pieces of data (difficult to do with document-based db or rdbms) example: neo4j

Document: represents atomic information

No way of joining/relating two documents using a docu-based db.

Order rows: part of aggregate root Order 🡺 save as a unit 🡺 some doc-dbs do not support transactions (mongo)

BISON: binary json

No limit to serializable type complexity (via JSON.Net)

ACID Transactions !!! (on write, on read by key, not the case for QUERIES)

Aggregation (map/reduce)

Transformations (e.g. query on a table and apply a view to resultset BEFORE it is returned to caller)

**Doc-based dbs encourage you to use cqrs:**

* Write aggregates (golden rule)
* Read viewmodels (depending on use case)

(protocol buffer)

Slide deck 02