Multi-model databases and the art of aircraft maintenance

Strata London 2015, 6 May 2015

Max Neunhöffer
About

- about me
  - Max Neunhöffer (@neunhoef) working for ArangoDB
  - Mathematician turned database engineer

- about the talk
  - multi-model databases, polyglot persistence
  - a case study in aircraft fleet management
  - it is real, but secret, some guesswork on my side
  - ArangoDB
An Aircraft Fleet
A single Aircraft
Or rather: a single Aircraft

Consists of several million parts.
A component: an Engine, ...
A hierarchy of thingies

- Fleet
  - Aircraft
    - Engine
      - Components
        - Part
Data

We have to store a lot of data for each item:

- names, type number, serial number, manufacturer
- maintenance intervals, maintenance dates, subcontractor
- links to manuals and documentation, contact persons
- warranty information, service contract information
- etc.
We have lots of different questions about this data:

- Find all parts in a component.
- Given a (broken) part, what is the smallest enclosing component for which there is a maintenance procedure?
- Which parts of this aircraft need maintenance next week?
- Find all components from a given supplier.
- etc.
A document store stores a set of documents, which means JSON data, these sets are called collections. The database has access to the contents of the documents.

- schema-less
- very versatile
Opaque values, only key lookup without secondary indexes:

\[ \Rightarrow \text{high performance and perfect scalability} \]

- more restricted queries — better scalability
Graph database

A graph database stores a labelled graph. Vertices and edges can be documents. Graphs are good to model relations.

- “graphy queries” like traversals are crucial
Polyglot Persistence

Idea

Use the right data model for each part of a system.

Take scalability needs into account!
A typical Use Case — an Online Shop

We need to hold

- **customer** data: usually homogeneous, but still variations
  - MySQL
- **product** data: even for a specialised business quite inhomogeneous
  - MongoDB
- **shopping carts**: need very fast lookup by session key
  - Redis
- **order and sales** data: relate customers and products
  - MongoDB
- **recommendation engine** data: links between different entities
  - Neo4j
Polyglot Persistence is nice, but …

Disadvantages

Consequence: One needs *multiple database systems* in the persistence layer of a *single* project!

Wouldn’t it be nice, …

… to enjoy the *benefits* without the *disadvantages*?
The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store, with a common query language for all three data models.

Important:

- is able to compete with specialised products on their turf
- allows for polyglot persistence using a single database
A multi-model data modeling approach

Idea

We store all data as documents. Since vertices and edges of graphs are documents, this allows to mix all three data models.

- **One document** (a vertex) for
  - the fleet,
  - each aircraft, (in different vertex collections).
  - each component, and
  - each part

- **Containment** is stored via edges (an item points to those contained).

- Use **document queries** where the graph structure is irrelevant.
- Use **graphy queries** when containment of items matters.
- Can mix the two within a single query.
A multi-model data modeling approach

Example: An aircraft

```json
{
  _key: "No18",
  kind: "aircraft",
  type: "747-800",
  manufacturer: "Boeing",
  built: "2001-07-07_12:12",
  lastMaintenance: "2015-05-04",
  nextMaintenance: "2015-05-07",
  flightHours: 1765,
  serialNo: "123456-78-9a",
  registration: "DK67BG",
  isMaintainable: true
}
```
A multi-model data modeling approach

Example: An engine

```json
{
    _key: "Engine765",
    kind: "component",
    type: "X67-12",
    manufacturer: "Rolls-Royce",
    built: "2001-05-17_09:23",
    nextMaintenance: "2015-06-01",
    lastMaintenance: "2015-05-04",
    flightHours: 812,
    serialNo: "987654-32-1a",
    fuelConsumption: 75.6,
    isMaintainable: true
}
```
A multi-model data modeling approach

Example: A screw

```json
{
  _key: "Screw56743",
  kind: "part",
  type: "S6L65Q1",
  material: "steel",
  manufacturer: "Fischer",
  serialNo: "546372635251",
  batch: "B5876a"
}
```
A multi-model data modeling approach

Example: An edge in the graph

```
{
    "_key": "E5364",
    "_from": "aircraft/No18",
    "_to": "components/Engine765",
    "kind": "contains"
}
```
A multi-model data modeling approach
Query time ...
Query time ...
Query time...
Query time ...
Find whole subtree

```
RETURN GRAPH_TRAVERSAL("FleetGraph", "components/Engine765", "outbound")
```

Returns all subcomponents and subparts of Engine765.
Find shortest path

\[
\text{RETURN GRAPH\_SHORTEST\_PATH("FleetGraph", "parts/Screw56744", }
\{\text{isMaintainable: true},
\{\text{direction: "inbound"}}\})
\]

Climbs „up“ from Screw56744 until a maintainable component is found.
FOR c IN components
  FILTER c.nextMaintenance <= "2015-05-15"
RETURN {key: c._key, nextMaintenance: c.nextMaintenance}

Disregards graph structure, finds all components with maintenance due.
A mix of them all

FOR p IN parts
    FILTER p.nextMaintenance <= "2015-05-15"
    LET path = GRAPH_SHORTEST_PATH("FleetGraph", p._id,
    {isMaintainable: true},
    {direction: "inbound"})

    LET c = DOCUMENT(path[0].vertex)
    FOR person IN contacts
        FILTER person._key == c.contact
        RETURN {part: p._id, component: c, contact: person}

Find parts, their corresponding maintenance component and join a contact person.
Other use cases

- E-commerce system
- Enterprise hierarchies and rights management
- Social networks
- Version management
- Complex user-created data
- Workflow management
- Organisation systems
- Knowledge graphs

Observation
Use cases that benefit from multi-model are actually prevalent!
ArangoDB

- is a multi-model database (document store & graph database),
- is open source and free (Apache 2 license),
- offers convenient queries (via HTTP/REST and AQL),
- including joins between different collections,
- configurable consistency guarantees using transactions
- memory efficient by shape detection,
- API extensible by JS code in the Foxx Microservice Framework,
- is easy to use with web front end and good documentation,
- and enjoys good community as well as professional support.
Extensible through JavaScript

The Foxx Microservice Framework

Allows you to extend the HTTP/REST API by your own routes, which you implement in JavaScript running on the database server, with direct access to the C++ DB engine.

Unprecedented possibilities for data centric services:
- custom-made complex queries or authorizations
- schema-validation
- push feeds, etc.
Distributed applications run well together on DCOSes like Mesosphere, Docker Swarm.

DCOS helps to build distributed apps (automatic failover, scaling).

ArangoDB's design lends itself well for Apache Mesos integration.

It is a win-win-cooperation.
https://www.arangodb.com

https://www.arangodb.com/foxx/

http://mesos.apache.org/

https://mesosphere.com/