Surviving Data in Large Doses

Tareq Abedrabbo

NoSQL Search Roadshow London 2013

About me

- CTO at OpenCredo
- Delivering large-scale data projects in a number of domains
- Co-author of Neo4j in Action (Manning)

What this talk is about…

Supermarkets

Meanwhile, in DevLand

Bob is an application developer

Bob wants to build an application.
Bob knows that a ~~relational database~~ is definitely not the right choice for his application

Bob chooses a **NoSQL** database because he likes it (he secretly thinks it's good for his CV too).

Bob goes for a three-tier architecture. It's separation of concerns. It's best practice.

Bob builds an object model first. It's Domain Driven Design. It's best practice.

Bob uses an object mapping framework. Databases should be hidden behind layers of abstraction. It's best practice.

Bob hopes for the best!

What challenges is Bob facing?

Suitability of the data model

Suitability of the architecture and the implementation

Ability to meet new requirements

Being able to use the selected technology to the best of its ability

Performance

A number of applications built on top of NoSQL technologies end up unfit for purpose

How did we get ourselves into such a mess?

- Technical evangelism
- Evolution in requirements
- Unthinking decisions
- Ill-informed opinions

Common problem: there is focus on **technology** and implementation, not on real **value**

So what's the alternative?

Separation of concerns based on data flow

Data flow

- Lifecycle
- Structure
- Size
- Velocity
- Purpose

How?

Identify the concerns: what do I care about?

Identify the **locality** of these concerns:
where are the natural boundaries?

Build focused

specialised models

Compose the models into a complete system

Computing is

data structures +

algorithms

If we accept that **separation of concerns** should be applied to **algorithms**, it is appropriate to apply the same thinking to **data**

The real value of this form of separation of concerns is true **decoupling**

What’s out there

CQRS

Polyglot Persistence

How do I apply it?

It depends on the data flow :)

For general-purpose data platforms, **micro services** work well

Build micro services that are closer to the natural underlying model

Other strategies are possible, for example if the data is highly volatile, consider **in-memory grids**

There are practical considerations - obviously

Don’t start with 10 different databases because you think you might eventually need all of them

How would that impact support and operations?

There is potential for simplification based on clearly targeted usage

Links

- Twitter: @tareq_abedrabbo
- Blog: http://www.terminalstate.net
- OpenCredo: http://www.opencredo.com

Thank you!