Storage Cost Evaluation and Sharding Analysis

for Denormalized NoSQL Schemas

Authors:

Mohamed Aymane Sfouli

George Shamieh

Luca Rougemont

Myriam Ait Said

ESILV – Big Data Structure

Introduction

This project focuses on evaluating the impact of different NoSQL denormalization strategies on storage cost and sharding distribution. Starting from a classical e-commerce model (Product, Category, Supplier, Stock, OrderLine, Client), we were asked to analyze five distinct denormalized database signatures (DB1–DB5) and quantify their consequences on storage usage.

To achieve this, we developed a **fully generic Python package** capable of interpreting any JSON Schema, computing the storage cost of documents, estimating database sizes, and evaluating sharding strategies.

The code is schema-driven, ensuring adaptability and robustness.

1 JSON Example and JSON Schema

1.1 Product Example

We created a detailed JSON example representing a product enriched with its categories and supplier. The example is intentionally different from the one seen in class, confirming the genericity of our code.

It includes descriptive fields, a price object, a list of categories, and structured supplier information.

1.2 Product JSON Schema

The JSON Schema enforces strict typing, required fields, nested objects, and structured arrays. It serves as the foundation for DB1–DB5.

2 JSON Schemas for DB1–DB5

The five database signatures correspond to different denormalization strategies:

- **DB1**: Product embeds its categories and supplier.
- **DB2**: Product additionally embeds stock information.
- **DB3**: Stock becomes the root and embeds a full Product.
- **DB4**: OrderLine becomes the root and embeds a full Product.
- **DB5**: Product embeds all related OrderLines.

3 Sizing Rules and Implementation

Storage sizing follows the lecture's rules:

Base value sizes

- Integer / Number: 8 bytes
- String: 80 bytes
- Date: 20 bytes
- LongString: 200 bytes

Key–value overhead

Each key-value pair costs an additional **12 bytes**.

Implementation Approach

A recursive function walks through the JSON Schema, computing sizes at each level. Collection sizes are then derived using cardinalities from `stats.json`. The result is a clear and reliable estimation pipeline.

4 Database Size Results

Database	Main Collection	Document Size (bytes)	Collection Size (GB)
DB1	Product	1096	0.1021
DB2	Product	1372	0.1278
DB3	Stock	1240	23.0968
DB4	OrderLine	1464	5453.8250
DB5	Product	1780	0.1658

Interpretation

DB1 and DB2 remain efficient. DB3 becomes heavy due to duplication. DB4 reaches a prohibitive size (5.4 TB). DB5 grows moderately and remains usable.

5 Sharding Analysis

Collection – Key	Avg Docs / Server	Avg Distinct Values / Server	Comment
St – #IDP	20,000	100	Good
St – #IDW	20,000	0.2	Very poor
OL – #IDC	4,000,000	1,000	Excellent
OL – #IDP	4,000,000	100	Acceptable
Prod – #IDP	100	100	Perfect
Prod – #brand	100	5	Moderate

6 Discussion on Dénormalization

Moderate denormalization (DB1–DB2) provides a balanced model. Aggressive embedding (DB3, DB4) results in massive redundancy. Modeling decisions must consider workload patterns and scalability constraints.

7 Conclusion

The implemented Python package meets all assignment requirements and offers a robust framework for analyzing NoSQL denormalization impacts. It demonstrates that:

- high-cardinality sharding keys (e.g., #IDC, #IDP) produce balanced distributions,
- extreme denormalization is rarely sustainable,
- schema-driven computation is essential for reusable data engineering workflows.