MongoDB in EEG/ERP Portal

Proof of Concept

Jakub Daněk (A12N0059P)

danekja@students.zcu.cz

Obsah

1 Project Background

The EEG/ERP Portal (further referenced as the Portal) is a java-based web application developed at the Department of Computer Science and Engineering at the University of West Bohemia. It's purpose is to store, manage, process and share neuroscience data. These data are often stored with lots of meta-data (environment, device settings, subject information). Currently the Portal's data layer is built on Oracle RDBMS. For the reasons mentioned further in this document, it's been decided to explore whether and how NOSQL databases (MongoDB in particular) could be used in the project.

Project Goals

In comparison to relational databases, the expected benefits from using MongoDB are:

- higher flexibility - ability to store various (even unexpected) meta-data without the need of changes in the application's data model.
- faster search by meta-data values - key:value search should provide performance boost thanks to removing the need to join tables

However, there are also several questions to discuss:

- What changes to the application would have to be done to switch the database engines?
- What disadvantages would the switch bring?

2 Implementation

2.1 Database

The following subset of experiment meta-data has been selected for the PoC:

- start and end time
- research group the experiment belongs to
  - its title and owner's name
- name of the experiment owner (the responsible person)
- experiment scenario
  - title, description, name of the scenario's owner
- subject information
  - name, date of birth, laterality, gender, age
- misc data
  - weather, environment notes, electrode settings, artifact information

Oracle

A copy of the Portal's database has been extended by additional triggers for auto-incrementing primary keys of the tables relevant to the experiment.

MongoDB

Sample JSON document has been designed to represent the default data structure. The document shall be attached to this paper.

2.2 Test Framework

A framework has been developed for easier and repeatable testing. It's written in Python which has been chosen for mature drivers for both the database engines, good handling of data collections and fast development. The framework provides following functionality:

- parameter-based generating of test data into Oracle DB
- export of all the experiments in OracleDB into MongoDB
- running bulks of queries on both the databases
  - measures the execution time
  - parameter-based bulk size
  - possibility to run bulks in parallel to increase server load

3 Results

3.1 Performance

Machine Specifications

CPU:	Intel i7 2.1 GHz, 4 cores
RAM:	8 GB
OS:	Gentoo Linux, kernel: 3.7.10
Oracle:	11g Release 2 (11.2)
MongoDB:	2.4.0

Tests

Three test cases (queries) were used:

1. Search by scenario name (single join in Oracle)
2. Search by name of a research group (two joins in Oracle)
3. Search by age of a subject (aggregation function in Oracle, date comparison in MongoDB)

Test iteration consisted of 100 queries in a row, which were run in parallel (in 1, 2, 4, 8, 16 and 32 threads). As a result 100 (200, 400, ..., 3200) queries had been run. For MongoDB six iterations were run without indexes, followed by six iterations with indexes created. For Oracle, all 12 iterations were run with indexes created for the relevant columns.

See attached tables for detailed results.

Tests have provided following output:

- MongoDB returns results on average 2-3 times faster than Oracle in most cases.
- Indexes in MongoDB didnt seem to have any significant effect on performance.
- When running queries in parallel MongoDB performs much better for queries 1. and 2.
  - Oracle kept the same query/s ratio (which resulted in exponential growth of test duration, depending on number of threads)
  - query/s ratio of MongoDB was increasing significantly up to 8 threads (and it kept the ratio for 16 and 32 threads)
- MongoDB had problems with query 3. under heavy load (16 and 32 threads)

  - this was the only case where Oracle performed better
  - one of the reasons might be that while Oracle can get the age value from a date, MongoDB has to compare dates directly

- Stability
  - MongoDB gave stable results for all iterations
  - Oracle experienced significant performance jumps during iterations, for unknown reason (as a result the average query/s ratio contains rather high statistical error)

3.2 Flexibility

Due to its query system and document nature, MongoDB is suitable for searching experiment meta-data. However, it doesn't provide an ultimate solution for the Portal as there are pieces of information for which relational integrity is required (typically subject or owner information). MongoDB does support foreign keys, yet their usage would result in multiple queries (as it doesn't support joins).

Therefore several options occur, each with own difficulties to overcome:

1. Keep current solution
   - easy way to maintain data integrity
   - not a flexible data model
2. Transfer all data to MongoDB
   - flexible data model
   - many problems
     - either lots of duplicate data or references
     - probably some application changes required
   - fast read queries
     - possible need for multiple queries (in case of references usage) to retrieve all the data
3. Parallel use of both database engines
   - keep data which require relational integrity in Oracle
   - keep experiment-specific meta-data in MongoDB
   - most difficult to implement
     - probably most complicated changes to the application
     - how to query effitiently?
     - how to handle transactions?

The option 2 seems to be the least suitable for the project such as the Portal. Even though the NOSQL database provides good facility for storing neuroscience experiments, it is not a good choice for current version of the Portal's domain. Any attempt to use solely MongoDB as the Portal's data storage would probably result in large changes in the application and its domain.

The option 1 is the safest one, while option 3 is worth further investigation as it might result in a solution which would be able to combine strong sides of the both concepts. The recommended approach would be creating a layer between the application and persistence - JPA (Hibernate) and MongoDB API. At the moment it is obvious that it's not going to be a trivial task.