Sensor-based Database with SensLog: A Case Study of SQL to NoSQL Migration

Prasoon Dadhich[1], Andrey Sadovykh[1], Alessandra Bagnato[1], Michal Kepka[2], Ondřej Kaas[2] and Karel Charvát[3]

[1]*Softeam Group, Paris, France*

[2]*University of West Bohemia, Plzeň, Czech Republic*

[3]*Lesprojekt-služby Ltd., Martinov, Czech Republic*

Keywords: NoSQL, Performance Testing, Modeling, Data Migration, Benchmarking, Sensor-based Database, Cloud Computing, Virtualization, Scaling, Data Architecture and Analysis.

Abstract: Sensors gained a significant role in the Internet of Things (IoT) applications in various industry sectors. The information retrieved from the sensors are generally stored in the database for post-processing and analysis. This sensor database could grow rapidly when the data is frequently collected by several sensors altogether. It is thus often required to scale databases as the volume of data increases dramatically. Cloud computing and new database technologies has become key technologies to solve these problems. Traditionally relational SQL databases are widely used and have proved reliable over time. However, the scalability of SQL databases at large scale has always been an issue. With the ever-growing data volumes, various new database technologies have appeared which proposes performance and scalability gains under severe conditions. They have often named as NoSQL databases as opposed to SQL databases. One of the challenges that have arisen is knowing how and when to migrate existing relational databases to NoSQL databases for performance and scalability. In the current paper, we present a work in progress with the DataBio project for the SensLog application case study with some initial success. We will report on the ideas and the migration approach of SensLog platform and the performance benchmarking.

1 INTRODUCTION

The growth of the IoT domain has increased importance for processing a large amount of sensor data. Sensor-based databases are a particular type of database with a constant *predictable data flow*(Kepka et al., 2017) in a fixed interval of time. The data flow is generally fixed with the sensors which are then attached to the units. These units are physical entities, for instance: a tractor in the field, field node with several connected sensors or a smart-phone (Řezník et al., 2016),(Kepka M., 2012). Sensor networks are widely used in several surveillance applications. These applications help us get the useful data for post-processing and analysis.

1.1 SensLog Case Study

SensLog is web-based sensor data management system suitable to process data both from static and mobile sensors. The SensLog application consists of server-side part and database part. Raw measurements from sensors are taken to gateways(Kepka et al., 2017). These gateways send the flow of measurements to the database with the SensLog API. SensLog database schema was designed and developed during recent years with features respecting standard `ISO 19156` (Probst, 2008). With the help of SensLog team, we have observed SensLog sensor-based data model and storage (Kepka et al., 2013), which, as in traditional data-intensive applications, relies on the relational database. The current implementation of SensLog data model in PostgreSQL database system (Pos, 2018) seems to be still effective. However, SensLog in future needs a much larger deployment and thus preparing for the infrastructure capable of dealing with big data. Therefore, this study will investigate the applicability of document-oriented databases for the future growth of SensLog.

Dadhich, P., Sadovykh, A., Bagnato, A., Kepka, M., Kaas, O. and Charvát, K.
Sensor-based Database with SensLog: A Case Study of SQL to NoSQL Migration.
DOI: 10.5220/0006909202390244
In *Proceedings of the 7th International Conference on Data Science, Technology and Applications (DATA 2018)*, pages 239-244
ISBN: 978-989-758-318-6

1.2 Objectives

Over the past years, Softeam research team has conducted several studies on the methods for model-driven design of data-intensive applications (Mar, 2018; da Silva et al., 2014; Aurélio Almeida da Silva and Sadovykh, 2014). Part of this work is the migration problem to help the architects and software engineers to move easier from one data storage technology to another with automated tools. In the context of the DataBio project(Dat, 2018) which focuses on data-driven bioeconomy, we develop the model-driven NoSQL migration tools in Modelio (Mod, 2018). Our major hypothesis is that the migration process should start with analysis and denormalization of relational structures (Mar, 2018). The next steps are the transformation to NoSQL (e.g. MongoDB) meta-model and deployment. Finally, an extensive testing should prove the benefits of the migration process comparing to the legacy application.

In this paper, we manually proceed with the above-mentioned process for our case study with SensLog. We concentrate on the performance testing part in order to prove the promised benefits which justifies further development of our methods in Modelio modeling tool.

2 BACKGROUND

In the world of software engineering, we have seen continuous changes in languages, platforms, and architectures. However throughout relational database has been the default choice for serious data storage in the industry context. With the growing data volumes, the need to integrate data from multiple sources and continuous strive for scale, there came new challenges in data storage. Thus, new database technologies such as MongoDB (Mon, 2018b), Cassandra (Cas, 2018), Neo4j(Neo, 2018) and Azure table storage (Azu, 2018) came into existence. NoSQL databases provided the advantage of building systems (Pokorny, 2013) that were more performing, scaled much better and were easier to program for.

Various proof of concepts and surveys have shown benefits to apply NoSQL databases(Pokorny, 2013). It has been proven particularly advantageous for the databases which do not rely much on the relations and/or could be denormalized with some measures(Pokorny, 2013; Dhanachandran, 2012). One of the promising cases in this respect could be the sensor industry.

Working with *SensLog team* who specializes in their tool for processing sensor queries, we furthermore explore the idea of processing sensor queries with the warehousing approach that relies on a document-oriented NoSQL database(Balazinska et al., 2007). In this approach, we principally have a data lake and an analytics database. The data lake is populated by the sensor gateway, which stores the sensor data in a predefined way for all the insertions. Then, this data could be transformed into a form that is easier for analytics application. The analytics database is usually smaller, centrally stored for faster read access with `SELECT` queries on specific indicators. In the meantime, the data lake containing the historical data can still be accessed with extraction mechanisms that may take longer time for the query. This approach enables acceptable response time for analytics data, while still allowing for complex query processing if needed through the data lake.

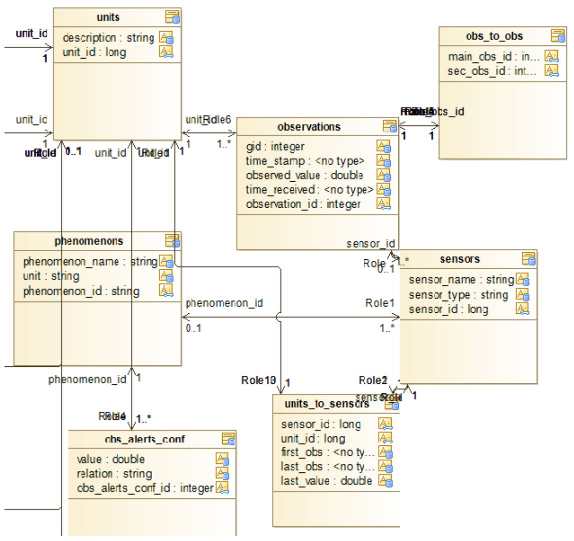


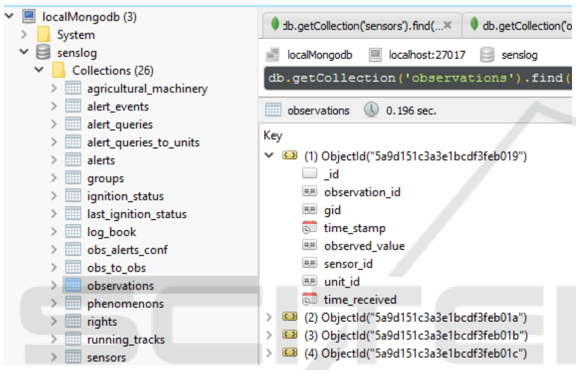
Figure 1: SensLog relational Schema observed with Modelio SQL designer.

3 MIGRATION

The current SensLog application is implemented with a PostgreSQL database. The SensLog application encounters certain performance limitations and our goal was to understand the bottlenecks and propose a more efficient architecture utilizing NoSQL concepts. Therefore, to start developing proof of concept, we had to explore the existing relational schema of the current SensLog application. We chose to translate the SQL schema into a data model using Modelio SQL Designer (SQL, 2018).

Our working hypothesis is denormalizing the data model in order to map it to NoSQL (Mar, 2018). Therefore, after observing complex relations among the tables in the SensLog data model, we investigated certain methods for denormalizing those tables towards getting to the MongoDB(Mon, 2018b) database. The main idea is to achieve a state where tables can be transformed into related documents and mapped to MongoDB. The denormalization methods are discussed in (Mar, 2018).

We applied methods starting with changing the connections to IDs to substitute relations among the MongoDB collections. In the second iteration, we substituted joints by embedding certain tables(Kanade et al., 2014). This method should be applied strategically depending on the amount of data to be duplicated. The good indication is the complexity of joint relations that can be observed with Modelio SQL Designer.



```
localMongodb (3)
  System
  senslog
    Collections (26)
      agricultural_machinery
      alert_events
      alert_queries
      alert_queries_to_units
      alerts
      groups
      ignition_status
      last_ignition_status
      log_book
      obs_alerts_conf
      obs_to_obs
      observations
      phenomenons
      rights
      running_tracks
      sensors
```

```
db.getCollection('sensors').find(...    db.getCollection('o
localMongodb    localhost:27017    senslog
db.getCollection('observations').find(
observations    0.196 sec.
Key
(1) ObjectId("5a9d151c3a3e1bcdf3feb019")
    _id
    observation_id
    gid
    time_stamp
    observed_value
    sensor_id
    unit_id
    time_received
(2) ObjectId("5a9d151c3a3e1bcdf3feb01a")
(3) ObjectId("5a9d151c3a3e1bcdf3feb01b")
(4) ObjectId("5a9d151c3a3e1bcdf3feb01c")
```

Figure 2: MongoDB migrated database snapshot from Robo3T interface.

Once deciding to migrate we used a specific Mongify tool (Mon, 2018a) where we could script these denormalization rules and translate the relational schema to a MongoDB database structure. That way, we could instrument the methods for ID replacement and embedding tables (Jadhav, 2017). The `figure 2` gives a snapshot of tables achieved after migration as seen in the Robo3T.

4 EVALUATING RESULTS

Once both systems, the legacy one and migrated one, were ready, we could proceed with the performance testing and compare both database systems. To achieve this we needed to set up a testbed consisting of two identical cloud machines hosting PostgreSQL and MongoDB. On both machines, we set up the SensLog data models and deployed the historical data.

With these two machines we organized the test plans that included the following scenarios:

- Throughput on insertion query.
- Response Time on insertion query.
- Throughput and Response Time with varying number of users varying from 10 to 30 as discussed with the SensLog team.
- Throughput and Response Time with specific queries.
- Throughput and Response Time with specific schema set.

Considering the limits of infrastructure, we concentrated our experiments with micro cloud machines. Choosing micro machines with small capacity of storage instead of bigger ones helped to demonstrate and learn the physical limits of stressing. That way, we could observe the behavior of the data insertion under stress conditions.

4.1 Testbed

In order to compare the results on the PostgreSQL and the MongoDB NoSQL SensLog data models, we deployed two identical Amazon instance machines with MongoDB and PostgreSQL schema respectively. We used two identical t2.micro Ubuntu machines with the following specifications:

Table 1: Amazon instance characteristics.

Amazon instance type	t2.micro
#Cores	1
#Threads	1
Instance storage	EBS Only
Operating System	Ubuntu 14.0

The first machine is deployed with the PostgreSQL version 9.6 with the initial schema received from SensLog team. The second machine is deployed with the MongoDB version 3.2 with the data model that we obtained in the migration process using Modelio SQL Designer and Mongify tool.

4.2 Performance Testing

The graphs below illustrate the results achieved with JMeter(Jme, 2018) performance testing tool. Those graphs compare the behavior of both PostgreSQL-based and MongoDB-base SensLog applications.

In this first set of experiments, we have concentrated on insertion testing (DataStax, 2018). We stressed both the machines by a test case involving a large number of users (i.e. sensors) to insert data each second in order to determine the maximum number of users to be served with a stable Response Time. We chose the most critical part of the data model in order to demonstrate the impact of the insertions on the system. In our context, the SensLog application heavily relies on the `Observation` table where all measurements are stored. Hence, we concentrated insertion queries on this table.

SensLog team targets that application is capable to handle *30 sensor data insertions per second* in a typical scenario. Thus, in the first part of our experiments, we measured the Response Time in this typical scenario on our testbed. We run test suite on both machines simultaneously and observed the difference. The results are shown in the snapshots below.

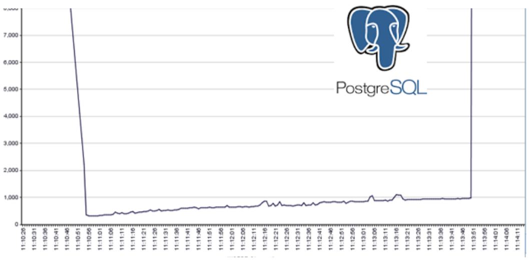


Figure 3: Observations Comparison Response Time Graph PostgreSQL.

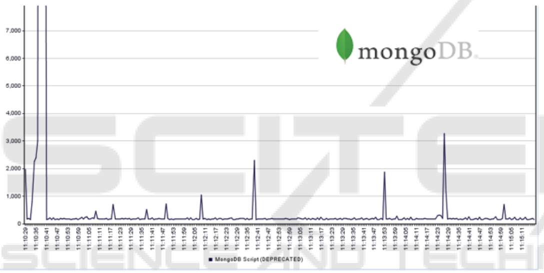


Figure 4: Observations Comparison Response Time Graph MongoDB.

We observe in the `figure 3` that it is going towards *exponential* curve whereas in `figure 4` we get *linear performance stability* in the NoSQL MongoDB insertions. In terms of throughput during insertion, we observed MongoDB performing *three times* more than the PostgreSQL machine.

In the second iteration, we decided to investigate in more details the MongoDB behavior by fine-tuning different parameters such as the number of users and the ramp-up time.

In the `figure 5` we observed a consistent Response Time when MongoDB was stressed with *15 users* for ramp-up time of 1 minute. There was a big jump after several insertions which is most likely because of the memory shortage in the small Amazon machine we have utilized in our proof of concept. From this lap, we can also see the behavior of MongoDB machine when it is near to getting full. That is a very appreciable behavior of a little jump in a still consistent way comparing to completely breaking up as in the PostgreSQL case.

In the next experiment, we did the same for *30 users* and observed similar behavior. Thus, with our experiments, we proved our hypothesis on linear consistency when using MongoDB.

The data insertions are done with the help of scripts in the JMeter tool (Jme, 2018) by generating relevant fake data as below.

PostgreSQL:

```
INSERT INTO public.observations(observation_id, gid,
time_stamp, observed_value, sensor_id, unit_id,
time_received)
VALUES (
${count_value},
${count_value},
${__Random(2000,2017)}-${__Random(10,12)}
     -01 00:00:00,
${__Random(0,100000)},
${__Random(1000,10000)},
${count_value},
${__Random(2000,2017)}-${__Random(10,12)}
     -01 00:00:00));
```

MongoDB:

```
db.observations.insert(
{
    "observation_id" : ${count_value},
    "gid" : ${count_value},
    "time_stamp" : ISODate("${__Random(2000,2017)}-
    ${__Random(10,12)}-01T00:00:00Z"),
    "observed_value" : ${__Random(0,100000)},
    "sensor_id" : ${__Random(1000,10000)} ,
    "unit_id" : ${count_value} ,
    "time_received" : ISODate("2014-10-01T00:00:00Z")
    }
 )
```

5 CONCLUSION AND FUTURE WORK

The major point with the sensor database is the constant flow of data from the sensor networks. This flow

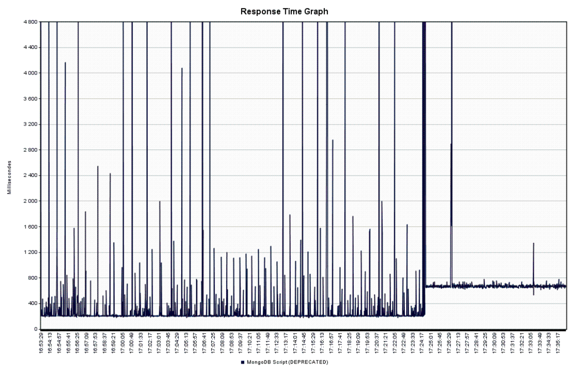


Figure 5: Observations Response Time Graph while Stressing with 15 users MongoDB.

if decoupled from the relational-based database, using the technique of data warehousing(Balazinska et al., 2007) can benefit the organization of the insertion in the database with respect to the Response Time. In SensLog database, we also observe the existing PostgreSQL heavily utilizing `SELECT` queries. MongoDB lacks support in `SELECT` queries and hence we weigh on preserving a part of the original architecture which will still have a centralized database operating with PostgreSQL.

From our tests, we observed SensLog Response Time graph in MongoDB to be growing linear compared to the equivalent PostgreSQL relational database. Therefore, we consider that a data warehouse on MongoDB will be a good solution for SensLog database, while still keeping the centralized database for the reporting system.

If there is any architecture decision to be taken for SensLog application, we would point to deploying MongoDB servers in the front receipting part to get direct data from the Sensors. We can later strategically utilize this database as a data pool for post-processing. Thus, our experimentations on the basis of proved concept with MongoDB sensor data model, we expect a performance optimization of threefold at the optimistic level in this architecture system.

This study is an important step forward for justifying our efforts for further research in the direction of automated tools for model-driven SQL to NoSQL migration with Modelio.

FUNDING ACKNOWLEDGEMENT

This work is under European research project DataBio (Dat, 2018) funded from European Union's Horizon 2020 research and innovative programme under grant agreement No. 732064.

REFERENCES

(2018). Azure. https://azure.microsoft.com/en-us/servi ces/storage/tables/. A NoSQL key-value store for rapid development using massive semi-structured datasets.

(2018). Cassandra. http://cassandra.apache.org/. Cassandra is an open-source distributed NoSQL database management system designed to handle large amounts of data across many commodity servers.

(2018). Databio. https://www.databio.eu/en/summary/. DataBio : Project to show the benefits of Big Data technologies in the raw material production from agriculture, forestry and fishery/aquaculture for the bioeconomy industry to produce food, energy and biomaterials responsibly and sustainably.

(2018). Jmeter. https://jmeter.apache.org/usermanual/i ndex.html. The Apache JMeter application is open source software, a pure Java application designed to load test functional behavior and measure performance.

(2018). Modelio. https://www.modelio.org/. The open source modeling tool environment.

(2018a). Mongify. http://mongify.com/. Mongify is a data translator system for moving your SQL data to MongoDB.

(2018b). Mongodb. https://www.mongodb.com/what-is-mongodb. Information about MongoDB.

(2018). Neo4j. https://neo4j.com/. Neo4j is a graph database management system developed by Neo4j.

(2018). Postgres. https://www.postgresql.org/. The implementation of PostgreSQL.

(2018). Sql to nosql migration. https://www.modelio.org/. An assessment of the migration from a relational database to a NoSQL store.

(2018). Sqldesigner. http://store.modelio.org/resource/modul es/sql-designer.html. A module in Modelio for Database modeling.

Aurélio Almeida da Silva, M. and Sadovykh, A. (2014). Multi-cloud and multi-data stores. In *Proceedings of the 4th International Conference on Cloud Computing and Services Science*, CLOSER 2014, pages 703–713, Portugal. SCITEPRESS - Science and Technology Publications, Lda.

Balazinska, M., Deshpande, A., Franklin, M. J., Gibbons, P. B., Gray, J., Hansen, M., Liebhold, M., Nath, S., Szalay, A., and Tao, V. (2007). Data management in the worldwide sensor web. *IEEE Pervasive Computing*, 6(2):30–40.

da Silva, M. A. A., Sadovykh, A., Bagnato, A., Cheptsov, A., and Adam, L. (2014). Juniper: Towards modeling approach enabling efficient platform for heterogeneous big data analysis. In *Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia*, CEE-SECR '14, pages 12:1–12:7, New York, NY, USA. ACM.

DataStax (2018). Benchmarking top nosql databases: Apache cassandra, couchbase, hbase, and mongodb.

Dhanachandran, S. (2012). Dynamic real time distributed sensor network based database management system using xml, java and php technologies. 4:9–20.

Jadhav, B. (2017). Gui for data migration and query conversion. *International Journal of Advanced Research in Computer Science and Software Engineering*.

Kanade, A., Gopal, A., and Kanade, S. (2014). A study of normalization and embedding in mongodb.

Kepka, M., Charvát, K., Šplíchal, M., Křivánek, Z., Musil, M., Leitgeb, Š., Kožuch, D., and Bērziņš, R. (2017). The senslog platform – a solution for sensors and citizen observatories. In Hřebíček, J., Denzer, R., Schimak, G., and Pitner, T., editors, *Environmental Software Systems. Computer Science for Environmental Protection*, pages 372–382, Cham. Springer International Publishing.

Kepka, M., Ježek, J., Charvat, K., and Musil, M. (2013). Complex solution for sensor network in precision farming.

Kepka M., J. J. (2012). Server-side solution for sensor data.

Pokorny, J. (2013). Nosql databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, 9(1):69–82.

Probst, F. (2008). Observations, measurements and semantic reference spaces. *Appl. Ontol.*, 3(1-2):63–89.

Řezník, T., Kepka, M., Charvat, K., Horakova, S., and Lukas, V. (2016). Challenges of agricultural monitoring: integration of the open farm management information system into geoss and digital earth. 34:012031.