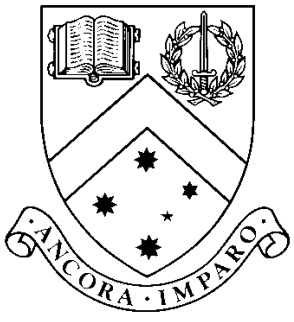

Monash University

Faculty of Information Technology

Computer Science Honours Reading Unit

Case Study of Inconsistent Railway Data With MongoDB

Author:
Jonathan Poltak Samosir

Supervisor:
Dr. Maria Indrawan-Santiago

May 18, 2015

Contents

1 Introduction

While relational database management systems (RDBMS) have been somewhat of a "go-to" solution for for a number of years for general data storage and management problems that many application developers face, we have noted a recent rise in the use of non-relational data management tools [8]. Most of these tools have traditionally fallen into the domain of big data analytics, with platforms such as the Hadoop ecosystem [1] being notably popular. However, outside of the domain of big data, looking more at general purpose data storage and management, what are now commonly referred to as "NoSQL" solutions are proving to be a popular solution.

NoSQL databases refer to those databases that are not built on top of the relational algebraic concepts, as laid out by Codd in 1970 [3], unlike the more commonly used RDBMS technologies, such as MySQL [2]. Being free of the strictness the relational model enforces on its data allows NoSQL databases to focus less on the overall structure of data, and more on factors such as scalability and performance [7].

While the relational model is a good fit for many data problems, its strictness in terms of flexibility of managing data eventually led to the introduction of the NoSQL model. The following characteristics can be given as a starting point for NoSQL databases in comparison to relational databases [6]:

- **Unstructured data support:** While the relational model would often force data to be stored in tabular formats, the NoSQL model does not force any kind of data schema.
- **Designed with distributed processing and horizontal scalability in mind:** Given the commoditisation of computer hardware in the last decade, support for horizontal scaling and processing among clusters is an important factor for adoption.
- **Less strict adherence to ACID principles:** While the relational model attempted to very much adhere to the transactional principles of data atomicity, consistency, isolation, and durability (ACID), this very much impacts performance in terms of distributed computing. Relaxing the strictness of adherence to these principles, allow many NoSQL databases to make the trade-off for higher performance.

Of course, these differences between the NoSQL model and relational model vary between each individual database technology's design, and trade-offs are often made depending on the goals and aims for that given database.

In this paper, we will look at the use of MongoDB [3], a popular NoSQL database solution, as a solution for a case study based upon a railway data problem using data from Monash University's Institute of Railway Technology (IRT). An overview of the case study in question will be given in §2. A small overview of the MongoDB database will be given in §3. Implementation and evaluation details will be given in §4 and §5, respectively, before concluding in §6.

[1] https://hadoop.apache.org/
[2] https://www.mysql.com/
[3] https://www.mongodb.org/

2 Case Study Overview

The case study that is being looked at involves a project that has been worked at at the Monash University Institute of Railway technology (IRT). The project involves trains the operate in the Pilbara region of Western Australia, taking ore and minerals from loading points at mines to a specified unloading port. The data that is being dealt with comes from numerous categories of sensors being placed on certain specialised train cars to record data monitoring track and train car conditions [4, 5].

Data currently gets unloaded at sent back in large batches to remote servers once the car completes a trip and pulls back into port [10]. The project currently makes use of a RDBMS solution to manage data storage, however this solution is faced with many problems that currently require painful work-arounds. The most obvious of which involves unreliable data being received from sensors. For example, in the case of a damaged or failed sensor, reliable data cannot be guaranteed to be returned from such a sensor. Hence, data received by the remote server is often inconsistently structured, and thus the data has to go through a series of preprocessing work arounds to fit in within the strict schema that the RDBMS expects. As the only guaranteed consistency in the data that gets received is the timestamp and geocoordinates, the strictly structured relational model is not an appropriate solution.

The Monash IRT team are currently investigating further solutions in the NoSQL big data space, where they intend to replace the current system with an appropriately tested solution. While the scope of that project is much larger than what will be covered in this paper, we will look at the differences in what is possible when attempting to use a NoSQL database for the data storage and management.

For this paper, we will be looking at the possibility of using MongoDB, a popular NoSQL database that does away with the concept of the schemas and tables so commonly found in databases following the relational model. This is done with the expectation of better handling of the given inconsistent data.

3 MongoDB Overview

Note that most of the information relevant to MongoDB in this section is sourced from the official MongoDB manual. [2]

3.1 Data storage

MongoDB can be described as a documented-oriented NoSQL database, which does not rely on schemas or tables to structure its data [9]. Data within Mongo is stored using JSON-like structured documents, called BSON [1], which allows for strongly typed data fields and binary data. BSON documents are then stored within "collections" which reside in a given "database" [9]. To relate back to the relational model, these concepts within Mongo can be compared as shown in Table 1.

Table 1: MongoDB conceptual structures as they relate to the relational model concepts.

MongoDB	**Relational Model**
Database	Database
Collection	Table
BSON Document	Record/Row/Tuple
BSON Key	Column/Field
BSON Value	Value

While these concepts within Mongo can be related back to concepts within the relational model for ease of understanding, they are not exactly the same. For example, a table in the relational model is structured in such a way that all records within that table must adhere to. Conversely, in a Mongo collection, several differently structured BSON documents may coexist freely. BSON documents generally follow some sort of predefined structure for that collection, to maintain data structure sanity, for the user's sake, however none of this is enforced.

This sort of freedom in terms of structure allows for similarly structured documents in database collections, where differences may only be found in that certain documents may miss keys that are present in other documents. This is highly rational in the way that, depending on the use-case, data may not be present for certain fields all the time. Thus instead of placing null or default values for missing data, Mongo allows the user to just leave them out of the relevant document. Note that document structure in Mongo is also dynamic in the way that it can be changed at any time; BSON keys and values may be added or removed from existing documents at any time.

3.2 Data interaction

MongoDB allows a lot of the similar type functionality in queries and data manipulation that is also expected in relational databases. Generally interactions with data are performed through JavaScript functions on collections, or cursors (subsets of documents within a collection), allowing general querying operations through use of the `find()` function, and standard CRUD operations through the `create()`, `update()`, and `remove()` functions.

MongoDB officially provides support for many drivers allowing interaction with Mongo databases in different programming languages, including JavaScript, Python, and Java. As well as providing drivers, MongoDB also provides a default shell, called the *mongo shell*, which allows general JavaScript code to be written to interact with the standard MongoDB API and perform ad-hoc queries. No official graphical user interface exists to

interact with MongoDB, however there are numerous third party clients, which essentially wrap around mongo shell.

MongoDB also supports more advanced data manipulation operations through use of aggregation functions. Generally, aggregation functions are user defined operations to perform on documents in collections to return results. These include general aggregate operation models, such as MapReduce, and often involve grouping documents by given keys and various queries to limit the amount of documents to process.

3.3 Further database features

3.3.1 Lack of joins

MongoDB, like many other NoSQL databases, does not support the joins of multiple tables or, in MongoDB's case, documents, as is common in relational databases. Instead, MongoDB allows two main ways of relating different documents to each other. These are via referencing and nested documents. Nested documents are the main recommended way of join multiple documents, unless the documents have many-to-many relationships. To allow nested documents, a BSON value can store a whole other document, or arrays of documents. Example use-cases of where nested documents are appropriate include documents that have natural "child" documents that only it will ever need to reference. For example, a blog post document may have many nested comment documents as the comments are specific to a given blog post. For documents where many things may refer to it, it is recommended to use references via BSON values containing other document IDs.

3.3.2 Indexing

Indexing in MongoDB is supported in a similar ways that are present in relational databases. Mongo builds a B-Tree data structure on a particular collection's key, allowing all documents within that collection with that key to be accessed in $O(logn)$ time, as opposed to $O(n)$ time if unindexed. MongoDB also supports numerous types of indexes, such as single-key, multi-key, geospatial, and hashed indexes.

3.3.3 Replication

MongoDB allows replication and management of multiple copies of the same data for the purpose of increasing data redundancy and availability. Replication in Mongo is managed using a primary instance of data, with secondaries replicating the data in the primary using operation logs (*oplogs*) to mimic any operations on data that were performed in the primary. Replicas have fault tolerance in the way that if the primary fails, an election is started to promote any of the secondaries to become a primary. In general, a client will only interact with a primary, however operations can also be specified to be performed on specific secondaries. However, as replication data is asynchronously managed, any reads performed specifically on secondaries may not give up-to-date data.

3.3.4 Sharding

Sharding in MongoDB allows for horizontal scaling of a Mongo database, where a given database or collection is shared among different "shards", or multiple MongoDB instances, generally running on separate machines. Sharding is performed on an indexed document key, dividing documents containing that key based on either range or computed hashes. Ranged-based sharding is generally recommended where data is generally expected to be

accessed in some sort of sequence, while hash-based sharding offers a guaranteed random distribution of data across shards, and more balanced equally-sized shards.

4 Implementation

Implementation of the database for the given Monash IRT data was all performed using MongoDB version 3.0.3, the latest version as of 2015-05-18. Interaction with the MongoDB database was all performed using the mongo shell and corresponding official NodeJS MongoDB driver.

4.1 Overview of the data

The test data used for this case study was acquired from the Monash IRT team, and was an extract of real data they have received from the railway sensors. As only one set of data was given, simulated datasets based upon the extract have also been created to simulate the type of inconsistencies the IRT team are facing, including differently structured datasets and datasets with missing data (in the case of non-functional sensors).

The dataset received from the IRT team has the layout as shown in Table 2. It consists of 99999 data points recorded at different times. The data received in this dataset is all consistent and contains readings from all the sensors. As this is not the case in with real data over time, simulated data has been used as stated previously. The only guaranteed data to be received is the **Time**, **Lat**, and **Lon** sensor recordings.

Table 2: The different types of data received from the Monash University Institute of Railway Technology team.

Data name	**Possible values**	**Meaning**
Time	Floating point value	Time in milliseconds since...
Speed	Floating point value	Speed in km/h (?)
LoadEmpty	-1, 0	Boolean flag denoting whether or not load is present in train cars.
km	Floating point value	Amount of kilometres travelled since...
Lat	Floating point value from -90 to +90	Geographic latitude coordinate at given reading.
Lon	Floating point value from -180 to +180	Geographic longitude coordinate at given reading.
Track	Integer value from 1 to 213841	Given track in which the train is located.
SND1	Floating point value	?
SND2	Floating point value	?
SND3	Floating point value	?
SND4	Floating point value	?
CouplerForce	Floating point value	The amount of force couple detected in the train car.
LateralAccel	Floating point value	The amount of lateral acceleration detected in the train car.
AccLeft	Floating point value	The amount of acceleration detected by the left sensor.
AccRight	Floating point value	The amount of acceleration detected by the right sensor.
BounceFront	Floating point value	The amount of bounce detected at the front of the car.
BounceRear	Floating point value	The amount of bounce detected at the rear of the car.
RockFront	Floating point value	The amount of rock detected at the front of the car.
RockRear	Floating point value	The amount of rock detected at the front of the car.

5 Evaluation and Discussion

6 Conclusion

References

[1] bsonspec.org/spec.html. `http://bsonspec.org/spec.html`. (Visited on 2015-05-13).

[2] The mongodb 3.0 manual — mongodb manual 3.0.3. `http://docs.mongodb.org/manual/`. (Visited on 2015-05-13).

[3] Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM 13*, 6 (1970), 377–387.

[4] Darby, M., Alvarez, E., McLeod, J., Tew, G., and Crew, G. The development of an instrumented wagon for continuously monitoring track condition. In *AusRAIL PLUS 2003, 17-19 November 2003, Sydney, NSW, Australia* (2003).

[5] Darby, M., Alvarez, E., McLeod, J., Tew, G., Crew, G., et al. Track condition monitoring: the next generation. In *Proceedings of 9th International Heavy Haul Association Conference* (2005), vol. 1, pp. 1–1.

[6] Indrawan-Santiago, M. Database research: Are we at a crossroad? reflection on nosql. In *Network-Based Information Systems (NBiS), 2012 15th International Conference on* (2012), IEEE, pp. 45–51.

[7] Leavitt, N. Will nosql databases live up to their promise? *Computer 43*, 2 (2010), 12–14.

[8] Padhy, R. P., Patra, M. R., and Satapathy, S. C. Rdbms to nosql: reviewing some next-generation non-relational databases. *International Journal of Advanced Engineering Science and Technologies 11*, 1 (2011), 15–30.

[9] Parker, Z., Poe, S., and Vrbsky, S. V. Comparing nosql mongodb to an sql db. In *Proceedings of the 51st ACM Southeast Conference* (2013), ACM, p. 5.

[10] Thomas, S., Hardie, G., and Thompson, C. Taking the guesswork out of speed restriction. In *CORE 2012: Global Perspectives; Conference on railway engineering, 10-12 September 2012, Brisbane, Australia* (2012), Engineers Australia, p. 707.