Report on R&D
Performance analysis and benchmarking of different databases for robotic applications

Rubanraj Ravichandran*
Matrikel Nr.: 9029355

Master Studies in Autonomous Systems

Bonn-Rhein-Sieg University of Applied Sciences

Advisors:
Prof. Dr. Manfred Kaul†
Nico Huebel‡
Sebastian Blumenthal§

December 15, 2017

*rubanraj.ravichandran@smail.inf.h-brs.de
†Manfred.Kaul@h-brs.de
‡nico.huebel@kuleuven.be
§blumenthal@locomotec.com

Declaration

I, **Rubanraj Ravichandran**, confirm that the research work titled: **"Performance analysis and benchmarking of different databases for robotic applications"** was solely carried out by myself. Any reference to work done by any other person or institution or any material obtained from other sources have been duly cited and referenced.

Date	Rubanraj Ravichandran

Abstract

Robotic applications generate huge amount of data from sensors and sometimes data are stored locally in the robot and most of the times data has been discarded after extracting meaningful information from it. People in the robotic community store these sensor data in the form of text files, log files or ROS bags and use them in future for navigation, manipulation, fault diagnosis, etc,. But there are potential disadvantages in storing data locally and the way data are stored.

For example, autonomous mobile robots need to share their data with the other robots in the environment to solve tasks in a team. If the data is not stored in a central place, or the robots are not connected in a network, then it will be a bottle neck situation for robots to get other robots sensor data. Even though the data is stored and shared between group of robots, if the data are not organized properly with no querying capabilities, then developers have to consider writing their own solutions to filter the required data from the log files.

To address this issue, we could opt the existing solutions from available databases to handle robot sensor data. In this research work, a set of databases will be selected from different categories like RDBMS, NoSQL, Graph, and Column oriented, and qualitative and quantitative analysis will be conducted under different configurable test cases. The results from this research work will give the best practices for when to use which databases, which data model should be considered and, insights of best database architectures, replication strategies and conflict resolution policies.

Contents

List of Figures

1 Introduction

1.1 Motivation

Modern robotics applications consume and generate already lots of data. However, this will further increase with the rise of the ubiquitous sensing provided by IoT and Industry 4.0 technologies. Most of the time, perceived data from sensors has been processed for decision making and disposed after the use. In the future, applications will require the sharing of data for the coordination of fleets of robots among themselves and with sensors in their environment. Typically, these data comes from different sources and requires different treatment. It can roughly be split into two categories:

- Streams of raw sensor data: This data is typically produced at high frequency and can be small (like the "tick" of an encoder) or large (like a point cloud produced by a laser scanner).
- Meta-data: This is connecting different types of sensor data together with the information and knowledge required to interpret them (like which robot has produced that data using which sensor with which settings at which position, applying which algorithm while performing what task). E.g., a point cloud of a laser scanner is not very useful if it is unknown from which position it was taken with which type of laser scanner and which settings.

As mentioned in these papers [1, 3, 4, 5], databases used in robotics for logging system, SLAM, storing sensor data for fault analysis and performance evaluation. But, the questions are which database is fit for which task, how efficiently we can use Relational database, NoSQL[6], Graph database[7], Multi modal database for different tasks in robotics. This project focuses on qualitative analysis of a reasonable subset of currently available databases and benchmark their performance metrics under different scenarios.

1.2 Structure

- Section 2 briefly describes the background about the topics which are relevant to this work.
- Section 3 shows the related work which has been done earlier in the field of database in robotic applications and also highlights the mechanism they followed to evaluate databases.
- Section 4 ...

2 Background

This section gives brief description/definitions of concepts relevant for this research work with respect to distributed databases.

2.1 Data model

A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to properties of the real world entities. For instance, a data model may specify that the data element representing a car be composed of a number of other elements which, in turn, represent the color and size of the car and define its owner.[8]

In the context of DBMS, data model represents how the data need to be stored and accessed from the database. Every database designed with their own data models and users can adopt any of these data model according to the application requirement.

2.1.1 Type of data models

- Relational data model - Data stored in rows and columns in the form of tables.
- Graph data model - Data stored in the form of tree like structure.
- Document data model - Data stored as documents or semi-structured data.
- Column family data model - Data stored in columns and similar columns category can be grouped into column family.
- Multi model - Database that supports more than one data model (E.g, Graph, Document, Relational).

These data models are popular in terms of flexibility and most used in new generation databases, and there are also other well-known database models available, but less frequently used are Hierarchical model, Network model, Object-oriented database model, Object-relational model, Entity-relationship model.

2.2 Database schema

Database schema is a structure described in a formal language supported by the database management system (DBMS). The term ”schema” refers to the organization of data as a blueprint of how the database is constructed.[9]

2.3 Query Language

Query language is a mechanism to read/access stored data from the database. Each database should be provided with their own query language with additional data filtering capabilities. For example, Mysql database have SQL (Structured Query Language), Cassandra have CQL (Cassandra Query Language), ArangoDB have AQL (Arango Query Language).

2.4 Distributed database

In distributed database, all elements in a single database is replicated in multiple database in different locations. Even if one database fails, we can still access our data from other available database.

Advantages

- High availability.
- Scalability

Disadvantages

- High overhead in making changes in all database instance
- High cost in terms of buying additional servers

2.4.1 Availability

Availability is a dominant property in distributed database which represents, whenever an user request for a data, database should be always available to serve the request quickly without fail.

2.4.2 Scalability

Scalability means, a system should be capable of handling the varying traffic workload by enhancing its resources (CPU core, RAM, Memory) or adding additional systems to split the workload.

2.4.2.1 Vertical scalability

In vertical scaling, we increase hardware resources in a single database server to compensate the current traffic. But this will not be an optimal solution always, because if the workload reaches critical point there is a chance that database might fail and no users can write/read data into database. This scenario is called single point of failure.

2.4.2.2 Horizontal scalability

In horizontal scaling, we add multiple database instance and distribute data equally among them and route incoming write/read requests to different database using load balancer. But the important thing is, the data should be consistent in all the databases and by doing this we can achieve 100% consistency.

2.5 Immutable and Mutable objects

In object-oriented and functional programming, an immutable object (unchangeable object) is an object whose state cannot be modified after it is created. This is in contrast to a mutable object (changeable object), which can be modified after it is created. [10]

Choosing between immutable and mutable objects are completely dependent on the application. Let consider two different scenarios, first scenario with more frequently updating data in the database and second one with few/no updation of data. For scenario one, mutable objects are best because instead of creating new memory space during every update, change the value in the existing memory address. For scenario two, immutable objects are best because it will not allow the user to change the existing value in the memory, but instead store the updated value in new memory address and delete the old memory space. Immutable objects are strongly suggested to use to achieve high performance and data consistency.

2.6 Replication

Replication is a process of creating multiple copies of same data and storing them in different database instances to ensure the data availability and improve the system fault tolerant state. If same data is available in multiple geographical locations, users can query data from their nearest location and, it will minimize the network traffic and latency.

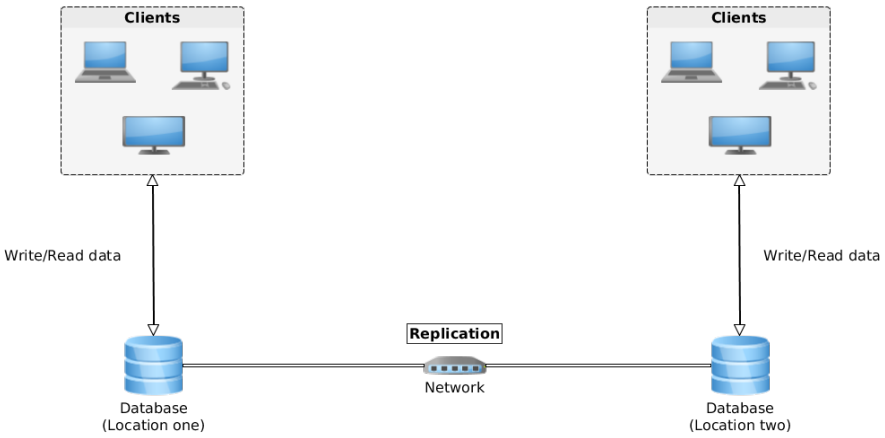

Figure 1: Generic Replication Architecture

2.6.1 Master-Slave Architecture

In master slave architecture, one database instance will act as a primary (master) node and all other database instances will act as a secondary (slave) nodes. Primary node can handle write and read requests, but slave nodes are restricted to accept write/update requests. It shows that, slave nodes are considered as backup nodes and if we have more slave nodes we can reduce the workload in master node.

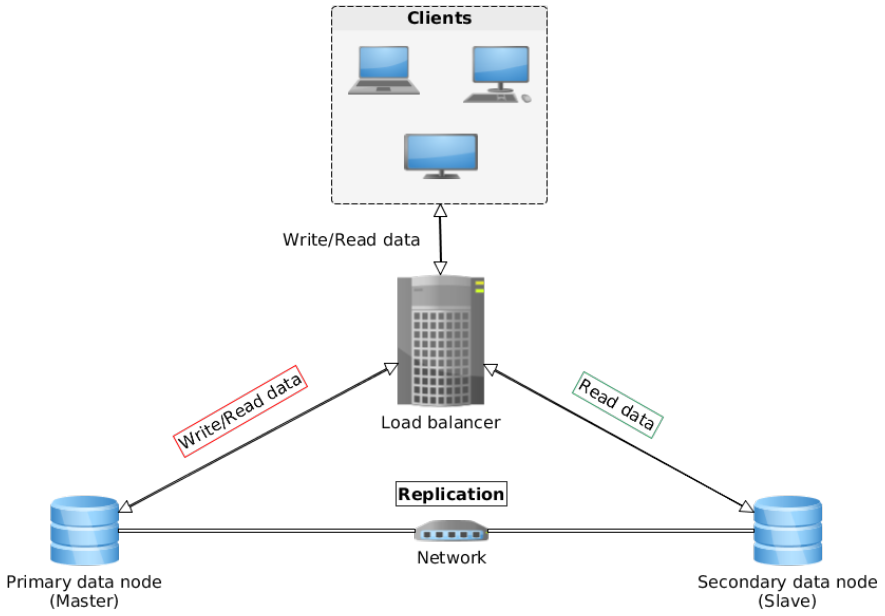

Figure 2: Master-Slave Replication Architecture

2.6.2 Master-Master Architecture

In master master architecture, all data nodes in the cluster will act as a master. Masters can be present in different physical location and any master can serve the client with its updated data. The advantages of using multi master replication are, high availability and fast server processing time. The disadvantages of multi master replication are, latency issues, loosely consistent (lazy and asynchronous) and violating ACID properties.[11]

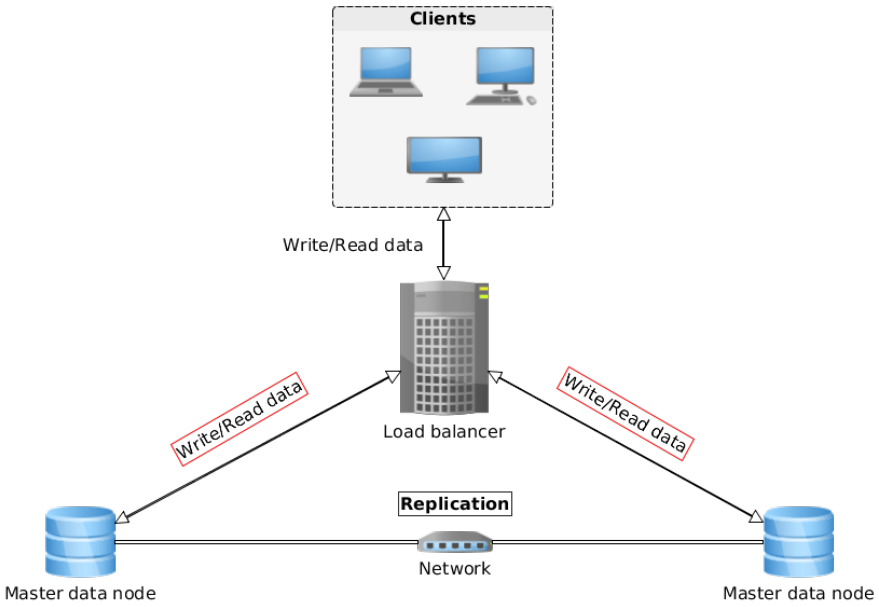

Figure 3: Master-Master Replication Architecture

2.7 CAP theorem

CAP theorem also called as Brewer's theorem named after computer scientist Eric Brewer[12], states that all distributed database systems can only provide two out of three properties which are shown in figure 4.

2.7.1 Consistency

Whenever user reads data from any database from the cluster, user will get the same version of data as a result. Consistency is achieved by updating several nodes before allowing further reads.

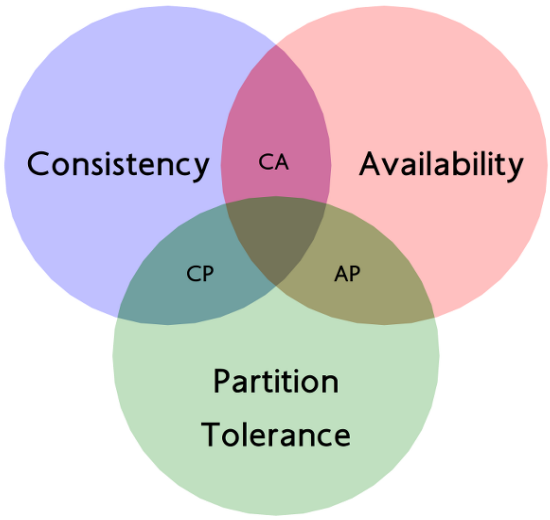

Figure 4: CAP Theorem [13]

2.7.2 Availability

All nodes in the distributed system should answer for user requests(write as well as read). Availability is achieved by replicating the data across different data node instances.

2.7.3 Partition tolerance

Here partition does not mean the disk/data partitioning, instead it means connection between the nodes partitioned (or disconnected) due to certain reasons like network failure, poor configuration, cable problem, etc.,

Distributed systems are somehow partitioned because of the problems which is mentioned above, but not so often. So, developers should keep partition tolerance in mind before developing systems. In partition tolerance mode, the system should choose either availability or consistency. How to pick anyone of these? It is simple, it depends on what our application required. If our application requires high availability, then the system should allow reads before updating all nodes and if application needs strong consistency, then the system should lock all nodes before allowing reads.

3 Related Works

People in robotics community already started to explore and use few database solutions to handle sensor data. Many works has been done prior related to evaluating databases and data formats in terms of robot applications[4, 14], comparison between SQL[2], NoSQL [15, 16], Graph databases[3], and use cases in robot applications where database could be used[1]. This section describes about the previous works related to databases used in robotic applications till now, comparison of different databases and also this section concludes why this research work is necessary than the existing work.

3.1 A generic robot database and its application in fault analysis and performance evaluation [1]

In this paper [1], a well known NoSQL database Cassandra has been integrated with Robotic Operating System(ROS) to handle the data in smart environments. Evaluated this approach in two scenarios, within a realistic robot exploration and with a pessimistic benchmark using randomly generated data. The final solution is compared with two other solutions that are commonly used in ROS applications(rosbag and mongo_ros). The research result shows that, Cassandra can handle terabytes of data and their time-stamp mechanism allows querying and retrieving data without additional efforts and also reveals that ROS Cassandra implementation consumes just a bit more memory than rosbag.

3.2 A performance comparison of SQL and NoSQL databases [2]

NoSQL databases claim that, it can outperform much better than current SQL(Structured Query Language) databases. The scope of this paper [2] is to analyze each NoSQL databases which are considered for evaluation with the SQL database. Comparison is done based on, Read, Write, Delete, and Instantiate operations on key-value stores. Additionally, investigated the performance of iterating over all keys and experimental results includes the time consumed by these operations. This paper highlighted key factors need to be considered for database comparison are,

- Scalability
- Consistency
- Support for data models

- Support for queries
- Management tools

Author also mentioned few more things need to be considered for database comparison from Indrawan santiago notes,

- Data model
- Transaction model
- Support for ad-hoc queries
- Indexing
- Shrading
- License type

There are five experiments conducted totally.

- In experiment one, NoSQL databases consumed less time compared to SQL to initiate the database bucket.
- In experiment two (Time taken to read values respective to provided keys), read performance is better in Couchbase, MongoDB and this results show that, the performance is not depend on the data model, because RavenDB is a document store like Couchbase and MongoDB, but RavenDB showed poor performance. MS SQL Express is a relational data model, but still it shows good performance than some other NoSQL data models.
- Experiment three (Write key-value pairs to the database and if the key value pair is already present in the database, then update the value else add a new pair) once again Couchbase and MongoDB shows good performance in writing. RavenDB and CouchDB is worse than that of SQL Express.
- Experiment four (Delete key value pairs from the bucket) Couchbase and MongoDB shows good performance in deleting. SQL Express is better than all other NoSQL database but Couchbase and MongoDB.
- Experiment five(Fetch all keys from the database) Couchbase have no API for fetching all keys, so it has been excluded for this experiment. SQL Express show good performance than all.

This paper considered almost all important NoSQL databases for comparison with SQL database. But the scenarios used for evaluation seems very simple and if we consider complex use cases in robotic applications like geographical range queries, text search based on dialogs and writing/reading blob files, this rankings and metrics may fail. Also this experiment conducted under single instance, so this performance results may differ in case of multi agent systems.

3.3 SLAMinDB: Centralized graph databases for mobile [3]

Robotic applications need memory recall mechanism for many tasks which includes localization, mapping, planning, visualization etc. Storing data in local system will isolate the data from the other robots. To achieve memory recall mechanism and sharing of data with other robots, author propose a shared centralized data persistence layer which is always available and it provides situational aware robot states. Also author mentions that, before storing the raw sensor data into database, local processing of sensor data in robot reduce the use of bandwidth. But local processing for huge amount data demands high processing unit and power source. For example, mobile robots need to calculate its position estimate continuously to localize itself. Two level database layer is implemented in the centralized server, to store simple data(odometric values) in graph database (Neo4j) and larger sensor data like RGB and depth image and laser scans in key value store (MongoDB). One of the benefits of graph database is, we can query and inference robots specific state at any point in the history.

Advantages of centralizing the data

- Multiple robots get access for long term.
- Reduced client side processing.
- Sharing of data and analysis between different sessions and robots.
- Aggregation and refinement of collective data.

Why relational database is not suitable for storing sensor data?

- It requires graph structure should be represented as tables.
- The schema should be defined during database creation.
- Large sensor data in the relational database will reduce the overall performance

The local process in robot communicate the relevant information back to the server for SLAM solver consumption. Query execution runs on the server side and send only the relevant information to client side. After receiving the recent pose estimate, robot can update the old value with new one and improve its own location estimate. For temporal queries, retrieving values from rosbag requires scanning of complete dataset. But in graph database, we can easily write queries to fetch specific data. As a result, for executing this query server took between 10 to 100 milliseconds. This paper shows robotic application developers a way to write procedural functions on the database to perform complex calculations.

3.4 Benchmarking of Relational and NoSQL Databases to Determine Constraints for Querying Robot Execution Logs [4]

This paper briefly explains the difficulties in handling the log messages from ROS(Robotic Operating System) and alternate ideas to store ROS logs for utilizing it in future. In ROS, logs are being stored in a flat file system called ROS bags. Using the bag files, we can only replay the tasks which has been recorded. But the disadvantages of ROS bags are, there is no capability to fetch data like database querying, and cannot do any arithmetic calculations on data like in other databases. Robots produce different types of sensor data, for example numerical values such as joint angles, transpose vectors, audio packets, etc and image like data such as laser scans, point clouds, stereo images, etc. To store these data formats, data store should have capability to querying which make it easy to analyze.

Three database systems (MongoDB, PostgreSQL, SQLite3) selected for evaluation and evaluated based on the following criteria.

- What is the maximum throughput of each database under loads consisting of varying numbers of topics, sizes of messages, and frequency of messages?
- Given a real world bag file (specifically, the MIT State dataset), how well does the database perform (e.g. percent of messages persisted) while running in a single machine configuration?
- How rich is the query interface supported by the database, and how fast can typical robot queries be executed?

Threaded database client, transfers the JSON data to the different databases which we considered for this experiment. For measuring throughput, 20 topics were generated for each synthetic data stream with each topic generating 500 messages. Each subscriber in the logging program limited to 10 messages in queue size, to avoid the message dropping. As frequencies and message size increases, messages will begin to drop when the database has reached it maximum throughput for inserts corresponding to messages of the given size. Message size were increased exponentially, for example the first message size is 5000 bytes, the second message size increased up to 0.64 MB. Data streams were generated at different frequencies, 4hz, 16hz and 64 hz.

Queries selected for query execution performance test are,

- Get the number of times the right gripper was closed.
- Get the every viewpoint position when the viewpoint was behind the robot.

- Get the number of times the right gripper was moved while in a particular region.
- Get the number of times a grasp was attempted with the right gripper when the camera was in front of the robot.

Result shows that, MongoDB performed the best across all queries requiring less than 50 milliseconds for each query 1,2,3 and 250 milliseconds for query 4. PostgreSQL is the next fastest across all queries and it is not designed for storing JSON data. Although SQLite3 had higher throughput on the synthetic dataset, MongoDB had the highest throughput on a real world dataset.

3.5 Conclusion

From all this approaches discussed above, they focused on very specific applications/settings and are always comparing only very few databases within this limited setting. Moreover, they benchmarked the performance with single robot in most of the cases. This benchmarking may not be helpful in multiple robot scenarios. Our intention of this work is, to go beyond that by comparing different model databases (SQL, NoSQL, Graph and Multi-model) in multiple settings/robots with the help of docker containers. This paper [3] seems to go in the direction of what we expect to be the best solution (combination of graph + data storage) and we will be further explored.

4 Problem formulation

The problem addressed in this work is, there is no specific databases available in the market for robotic applications and our task is to identify a set of databases and benchmark them to decide which database will be fit for robotic applications. When data about different robots and their environment combined together called as world model. These days, developers in robotics often use self-written, ad-hoc solutions for handling robotic data. However, in recent years many commercial as well as open-source databases became available that perform similar tasks with promising better performance (wrt features, performance, and querying) as well as additional features compared to the ad-hoc solutions from robotics. However, they all come with advantages and disadvantages and were not designed for robotic applications. So the robotic specific structures and data models need to be imposed on these databases to find out which databases performing better.

5 Approach

To address this problem, a set of Relational, NoSQL, Graph and multi-model databases will be compared qualitatively based on their features. For comparison, databases will be collected from state-of-the-art techniques and current top ranked databases. The selected features include common database criterias into account. Furthermore, features will be derived from the below robotic usecases. Benchmarking scenarios will be defined and implemented from robotic research projects(e.g. ROPOD). Finally, a selection of the databases will be constructed using docker containers and compared against the benchmarking scenarios. This comparison is based on the performance evaluation metrics(e.g. throughput, query execution time, message size, frequency, data replication time and number of robots involved in the scene). The result will be a suggestion for the best practice when to use which database in robotics.

References

[1] Tim Niemueller, Gerhard Lakemeyer, and Siddhartha S Srinivasa. A generic robot database and its application in fault analysis and performance evaluation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 364–369. IEEE, 2012.

[2] Y. Li and S. Manoharan. A performance comparison of sql and nosql databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 15–19, Aug 2013. doi: 10.1109/PACRIM.2013.6625441.

[3] D. Fourie, S. Claassens, S. Pillai, R. Mata, and J. Leonard. Slamindb: Centralized graph databases for mobile robotics. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6331–6337, May 2017. doi: 10.1109/ICRA.2017.7989749.

[4] Alexander J Fiannaca and Justin Huang. Benchmarking of relational and nosql databases to determine constraints for querying robot execution logs.

[5] André Dietrich, Siba Mohammad, Sebastian Zug, and Jörg Kaiser. Ros meets cassandra: Data management in smart environments with nosql.

[6] Robin Hecht and Stefan Jablonski. Nosql evaluation: A use case oriented survey. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 336–341. IEEE, 2011.

[7] R. Angles. A comparison of current graph database models. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 171–177, April 2012. doi: 10.1109/ICDEW.2012.31.

[8] wikipedia. Data model, 2017. URL https://en.wikipedia.org/wiki/Data_model/. [Online; accessed 10-December-2017].

[9] wikipedia. Database schema, 2017. URL https://en.wikipedia.org/wiki/Database_schema. [Online; accessed 10-December-2017].

[10] wikipedia. Immutable object, 2017. URL https://en.wikipedia.org/wiki/Immutable_object. [Online; accessed 10-December-2017].

[11] wikipedia. Master-master replication, 2017. URL https://en.wikipedia.org/wiki/Multi-master_replication. [Online; accessed 10-December-2017].

[12] wikipedia. Cap theorem, 2017. URL https://en.wikipedia.org/wiki/CAP_theorem. [Online; accessed 15-December-2017].

[13] stackchief. Cap theorem, 2017. URL https://www.stackchief.com/blog/CAP%20Theorem%20Explained. [Online; accessed 12-December-2017].

[14] André Dietrich, Sebastian Zug, Siba Mohammad, and Jörg Kaiser. Distributed management and representation of data and context in robotic applications. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1133–1140. IEEE, 2014.

[15] João Ricardo Lourenço, Bruno Cabral, Paulo Carreiro, Marco Vieira, and Jorge Bernardino. Choosing the right nosql database for the job: a quality attribute evaluation. *Journal of Big Data*, 2(1):18, 2015.

[16] B. G. Tudorica and C. Bucur. A comparison between several nosql databases with comments and notes. In *2011 RoEduNet International Conference 10th Edition: Networking in Education and Research*, pages 1–5, June 2011. doi: 10.1109/RoEduNet.2011.5993686.