Combining Relational and Semi-Structured Databases for an Inquiry Application

Marius Ebel and Martin Hulin

University of Applied Sciences Ravensburg-Weingarten, 88250 Weingarten, Germany
{ebelma,hulin}@hs-weingarten.de

Abstract. The popularity of NoSQL databases keeps growing and more companies have been moving away from relational databases to non-relational NoSQL databases. In this paper, the partitioning of a relational data model of an inquiry system into semi-structured and relational data for storage in both SQL and NoSQL databases is shown. Furthermore, the CAP theorem will be applied to categorize the storing of the correct parts of the model into their corresponding databases. As a result of these reorganizations and the introduction of additional histogram data, overall performance improvements of about 93% to 98% are achieved.

Keywords: NoSQL, Document Stores, MongoDB, semi-structured data, inquiry system

1 Introduction

Not Only SQL (NoSQL) – reintroduced by Evans in 2009 [7] – stands for a variety of new data stores which now are gaining population on the market [10]. Long time fully relational data stores have been sufficient for most purposes. But since companies and organizations have started to collect huge amounts of data, such as customer, sales and other data for further analysis, several types of non-relational data stores are preferred over the relational ones [10]. Traditional SQL databases come with the need of a fixed schema organized in tables, columns and rows, which cannot handle the changed needs for today's database applications.

NoSQL does not necessarily mean schema-free. There are also NoSQL databases for structured data like Google's Bigtable [4]. Among the schema-free NoSQL databases there are numerous data stores like the key-value-stores Redis [11] and the document stores like Amazon SimpleDB [15], Apache CouchDB and MongoDB. NoSQL data stores themselves provide a variety of sophisticated techniques like MapReduce [6] and better mechanisms for horizontal scalability [3] among others.

NoSQL document stores form only one group of this large variety of NoSQL data stores. *These databases store and organize data as collections of documents, rather than as structured tables with uniformsized fields for each record. With these databases, users can add any number of fields of any length to a document* [10, p. 13]. They organize data in—as the name indicates—documents. Documents are treated as objects with dynamic properties, which means that a

document can be interpreted as a dynamic list of key-value pairs. The approach described in this paper makes use of the absence of schemas in document-oriented NoSQL databases to store dynamic, semi-structured data.

SQL databases mostly use proprietary connection drivers to handle communication with the server, where data transfer is just a "black box". NoSQL databases are different in this case. Many of them use *Representational State Transfer* (REST) [8,14] where often a community-developed connection driver has to be used (Redis, Apache CouchDB), but some also come with a proprietary connection driver (MongoDB). In most projects there is either no time for developing a connection driver or it is unsafe to rely on community projects. Therefore, the presence of a proprietary connection driver can be a very important factor for choosing the appropriate NoSQL database.

The approach to use semi-structured data structures is basically not new. For instance, the database system CDS/ISIS is a type of semi-structured NoSQL database, which is used since the 1980's by a vast amount of academic libraries [13]. But the way of using semi-structured data described in the following section focuses on reorganizing previously relational data into semi-structured record sets and on the optimization of computational effort of statistical data.

Furthermore, we will show an approach to affect data organization through dynamic semi-structured data and the subsequent querying, which is also applied in the inquiry system INKIDU[1].

2 Methodology

The use case of the methods described in the sequel is the online inquiry platform INKIDU. INKIDU provides the user with the ability to design questionnaires, which can be statistically evaluated after completing the inquiry. Questionnaires in INKIDU are user-defined sequences of questions. Each question can be of a different type such as rating questions, single and multiple choice questions, free text questions, etc. After submitting a questionnaire, the submitted data has to be stored for further result analysis. A questionnaire therefore has two factors affecting the possible amount of data, which can be produced by each user: The number of questions and for each question its subsequent type. The effective structure and amount of answers of an answer set is finally determined by the questions, which the user decided to give an answer for. Hence, there is no way to predict the exact amount and structure of data, which will be produced by each user. Conclusively only a rough estimate of the structure is known: A list of key-value-pairs, i.e. the key is the question identifier and the value is the user-given answer. Considering the variable length of such a list and not knowing the data type of each value, only semi-structured data is given. In a system meant for strictly structured relational data, the presence of just semi-structured data leads to a conflicting situation of storing a set of semi-structured data with an unpredictable number of fields and data types.

[1] `http://www.inkidu.de/`

2.1 Current Data Organization

The current way of organizing answer sets is shown in Figure 1, where answer sets are organized vertically. An answer set containing all of an user's answers belonging to a questionnaire is split into tuples consisting of the mandatory primary key, the inquiry this answer belongs to, the answered question, and a field referencing the first primary key value of all answers belonging to the same answer set. The last field is created by the need of making the assignment of answers reconstructable, e.g. for data export and for allowing further statistical cross-analysis. As the data model shows, the redundant reference to the inquiry is not necessary, but is held to eliminate the need of a table join and therefore to keep query times at a reasonable level, when e.g. retrieving all answer sets belonging to a certain inquiry. These bottlenecks such as redundant foreign keys, complex queries and huge index data, arise from the RDBMS' need of a fixed table schema, which requires a predefined and therefore predictable structure of the data.

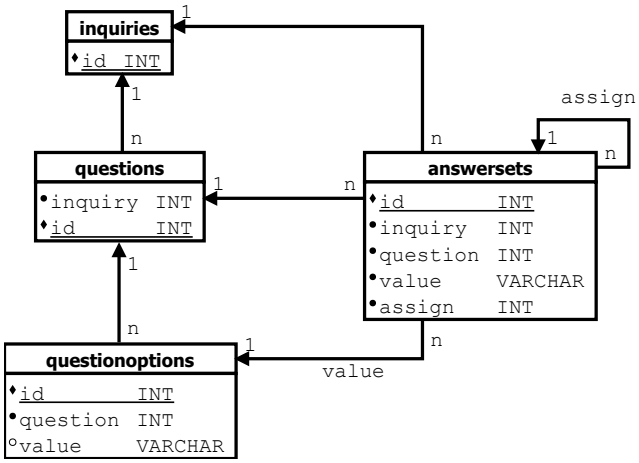

Fig. 1. Excerpt from the current fully relational data model of INKIDU being reorganized.

In order to demonstrate the current data organization and the approaches described in the following, a questionnaire with one rating question and one multiple choice question with three options is taken as a reference example. There are two answer sets to be shown in different manners of data organization:
`(rating=1, multiple choice=[option 1, option 2])`
`(rating=4, multiple choice=[option 1, option 2, option 3])`
Applying the fully relational approach on the reference example data, answer sets, shown in Table 2.1, are produced.

id	**inquiry**	**question**	**answers**	**assign**	***remarks***
3042397	18722	366519	1	3042397	rating value 1
3042400	18722	366519	4	3042400	rating value 4
3042398	18722	366520	203648	3042397	option 1
3042399	18722	366520	203649	3042397	option 2
3042401	18722	366520	203648	3042400	option 1
3042402	18722	366520	203649	3042400	option 2
3042403	18722	366520	203650	3042400	option 3

Table 1. Answer sets which result from storing the above described example data in the fully relational data model. The values of the column *answers* for *option 1 – 3* are foreign keys referencing record sets of the the table *questionoptions*.

2.2 Organizing Answer Sets

In the following, the approach of reorganizing the above data structure is described. With the emergence of schema-free databases that do not depend on a predefined structure of data records, the data reorganization can be done not only much more intuitively, but also more efficiently. This means that the construction of the data records requires less effort such as data redundancies and index data and produces a structure which is easy to read for machines and humans as well.

Semi-structured data is often explained as "schemaless" or "self-describing", terms that indicate that there is no separate description of the type or structure of data. Typically, when we store or program with a piece of data, we first describe the structure (type, schema) of that data and then create instances of that type (or populate) the schema. In semi-structured data we directly describe the data using a simple syntax [1, p. 11].

In this case we use *JavaScript Object Notation* (JSON) to describe an answer data set, e.g. in terms of NoSQL a document such as the following record sets, which result from the reference example data.

```
{ // first answer set:
  // (rating=1, multiple choice=[option 1, option 2])
  inquiry: 18722,
  data : {
    366519 : 1
    366520 : [ 203648, 203649 ]
  }
}

{ // second answer set:
  // (rating=4, multiple choice=[option 1, option 2, option 3])
  inquiry: 18722,
  data : {
```

```
    366519 : 4
    366520 : [ 203648, 203649, 203650 ]
  }
}
```

The use of JSON notation is proposed by MongoDB, which is used for this project. Binary JSON (BSON) is the format used by MongoDB to store documents and to transfer them via network [12,3]. As shown above an answer set, formerly to be divided into several records, can now be grouped into one single, semi-structured record, which uses a key-value list mapping question ids to the user-given answers. Hence, a convention of a basic loose structure of an answer set can be decribed as the following: a reference to the inquiry the user participated and n key-value pairs of the data mapping as described before.

The advantage of a non-structured NoSQL-Database is now clearly visible: All the data belonging to the same answer set can be kept in one single record that can be easily queried by the inquiry field and still does not need to fulfill the requirement of a predefined number of fields or data types. This way of storing data simplifies data management and is easier to understand.

2.3 Data Condensation

The approach described above can be refined to group all answer sets into a single record, i.e. to condense data from multiple record sets into a single one. The reference example data would result in a record set shown below.

```
{
  inquiry: 18722,
  data : [
    { // first answer set:
      // (rating=1, multiple choice=[option 1, option 2])
      366519 : 1
      366520 : [ 203648, 203649 ]
    },
    { // second answer set:
      // (rating=4, multiple choice=[option 1, option 2, option 3])
      366519 : 4
      366520 : [ 203648, 203649, 203650 ]
    }
  ]
}
```

Such a condensed record set can grow very large, which is a disadvantage especially for real-world applications. It increases the memory consumption of answer processing outside the database resulting in severe scalability problems

for the whole application. But with smaller amounts of data or less strict scalability requirements, such approach is still a better way of data modeling with comfortable handling.

Databases use B-Trees for indexing data [5]. The previously fully relational data model required three fields per answer record set (i.e. for every answer to every question) to be indexed in order to keep query times at a reasonable level. The B-Trees indexing only the answer sets of INKIDU consume more storage space than the data itself. Using the semi-structured approach the index data is minimized, because only one field per collection (*inquiry*) and less record sets have to be indexed. This is true whether the answer sets are stored in multiple records or are condensed to one record.

2.4 Histograms

The price to pay for less index data and easier queries is that some functions formerly used in SQL queries aren't available anymore. Examples for some of these functions are minimum/maximum, average, standard deviation, etc. Keeping in mind that an inquiry application also includes the statistical analysis of the retrieved answers, the absence of these functions is fatal. These functions have to be regained and unfortunately have to be implemented outside the database. Using the situation to our benefit, the computation of statistical data can be done more efficiently by introducing another semi-structured dynamic data structure, which is described in the following. For better understanding the formula of the arithmetic mean has to be considered first:

$$\bar{x}_{\text{naive}} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + x_3 + \ldots + x_i}{n} \tag{1}$$

This naive formula implies that every element $x \in X$ has to be processed for calculating $\bar{x}_{\text{naive}}$. This means that the computations of the arithmetic mean and the standard deviation result in severe scalability problems for larger or growing amounts of answers. The computational effort for e.g. the average grows linearly by $\mathcal{O}(n)$ with n as the number of record sets. Therefore these calculations have to be optimized, which is now done by storing additional data. The linear growth of computational effort by the number of record sets is—especially in the described scenario—a bottleneck. To tackle the challenge of reducing the computational effort, data is reorganized to histograms.

Per definition, a histogram is a graphical representation of the distribution of data. In the above described scenario, a histogram represents the distribution of answers to a single question holding the absolute frequency of every option. An option can be either a foreign key referencing a question option of a multiple/single choice question or a rating value of a rating question. Hence, the histogram can be represented as a set of option-frequency pairs, which can be used to do a much more efficient calculation of the arithmetic mean and the standard deviation. The following schema shows a representation of a histogram, which can be easily projected onto a semi-structured NoSQL document. In this schema

o stands for *option* and f stands for *frequency*, where a list of these option-frequency mappings is the actual data part of a histogram, $H_{\text{inquiry, question}}$, without the inquiry and question information.

$$\text{data}(H_{\text{inquiry, question}}) = \begin{bmatrix} o_1 \ , \ f_1 \\ o_2 \ , \ f_2 \\ o_3 \ , \ f_3 \\ \vdots \ , \ \vdots \\ o_k \ , \ f_k \end{bmatrix} \overset{\text{e.g.}}{=} \begin{bmatrix} 1 \ , \ 594 \\ 2 \ , \ 453 \\ 3 \ , \ 203 \\ 4 \ , \ 134 \\ 5 \ , \ 43 \end{bmatrix} \tag{2}$$

As shown in Eqns. 2+3, every option is "weighted" by its absolute frequency, which means that the set of data is not iterated over every record set, but over every question option instead. This makes the computational effort still grow linearly by $\mathcal{O}(k)$ where k is the number of available question options (revise following sentence), but rather by the number of available question answers than by the number of given record sets.

$$\bar{x}_{\text{hist}} = \frac{1}{n} \sum_{i=1}^{k} (o_i \cdot f_i) = \frac{o_1 f_1 + o_2 f_2 + o_3 f_3 + \ldots o_k f_k}{n} \tag{3}$$

The correctness of Eqn. 3 is given by the fact that the sum of all frequencies is equal to the number of given answers: $\sum_{i=1}^{k} f_i = n$. Now the number of different question options is a predefined number, which is independent from the growing number of incoming answers and therefore makes the computational effort predictable and as small as possible: It is independent of the number of users taking part in a questionnaire. Of course, the complete computation time for calculating the mean remains the same. However it is moved from a time critical part of the application (statistical analysis) to an uncritical part (storing new answer sets). Additionally the computational effort is distributed over every user submitting answers to the system, which results in shorter processing times for visualizing and displaying statistical analysis reports (see Section 3.3).

The disadvantage of storing redundant frequency values is justified by the performance gain of the statistical analysis, a core feature of inquiry applications. As mentioned above a document in terms of a document-oriented NoSQL database is a list of nestable key-value pairs. The semi-structure of a histogram data set now can be modeled as follows; consisting of a fixed and a dynamic part. The fixed part is the identifying header, which includes the inquiry id and the associated question id. The dynamic part is the actual mapping from question option onto absolute frequency.

3 Software Architecture

In this section the realization of the above concepts is described. The application is, as already mentioned, the online inquiry platform INKIDU, which allows the user to create and evaluate user-defined questionnaires.

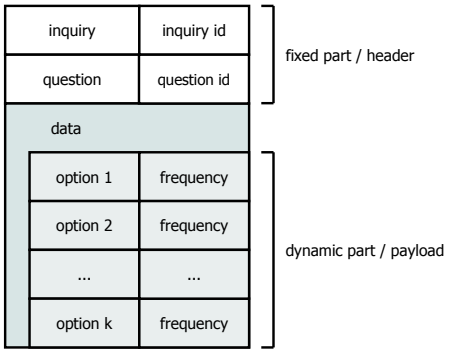

Fig. 2. Structure of a semi-structured histogram record. The upper part ("header") is the fixed part, which can be found in every histogram record. The lower part ("payload") is the dynamic part, which is variable in its number of key-value-pairs.

3.1 System Overview

The concept of integration in case of INKIDU includes the parallel usage of a traditional SQL database and a NoSQL document store. The NoSQL document store is provided by MongoDB[2] while MySQL [3] is used as the data store of the relational data model.

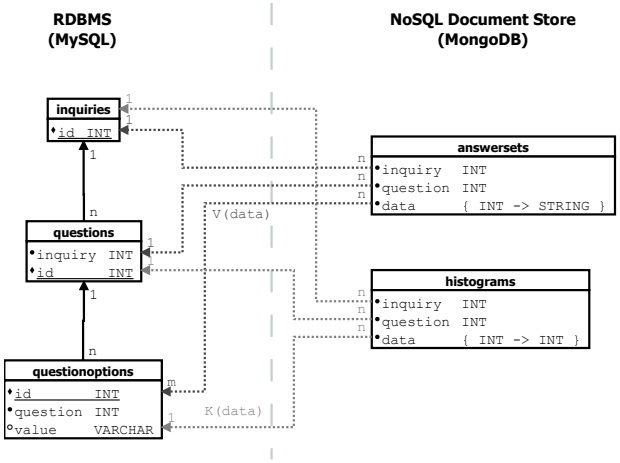

Fig. 3. Excerpt of the data model of INKIDU showing which parts belong to the relational and non-relational data model.

[2] http://www.mongodb.org/
[3] http://www.mysql.de/

Consistency and referential integrity are important features in relational data models. With the features of mass data storage, database scalability and high availability NoSQL databases often do not provide native support for ACID transactions. NoSQL databases are able to offer performance and scalability and keep strong consistency out of their functionalities, which is a problem for several types of applications [10]. For instance, in case of INKIDU there are parts of the data model which do require ACID transactions and strong consistency as well as parts which do not require this functionality. Therefore, as Figure 3 shows, the data model is distributed over the two different databases. So the decision to make it a hybrid system consisting of relational and non-relational databases is explained by two different factors. The first factor is, that the approach of increasing system performance by reorganizing data works also on single server setups. Other mechanisms increasing performance by scaling a system horizontally (e.g. load balancing) need a multiple server setup to work. The second factor is defined by the different requirements towards consistency and referential integrity. However, there are approaches to ensure consistency within a non-relational data model [16], which for now exceeds the resources of this project.

The hierarchy consisting of `inquiries`, `questions`, `questionoptions`, etc. has to be consistent at any time. This includes on one hand ACID transactions and on the other hand referential integrity, such that e.g. no orphaned records can exist. This means that the absence of foreign key checking and the presence of eventual consistency as achieved by MongoDB is not sufficient. Therefore MySQL is still used to store these structured data. NoSQL database management systems can store data, which change dynamically in size much better than relational ones; for instance nodes can be added dynamically to increase horizontal scalability. Considering the three guarantees *consistency, availability and partition-tolerance* of the CAP theorem [2,9], consistency and availability are most important for the relational part. Availability and partition-tolerance are most important for the non-relational of INKIDU due to the fact that it is not tolerable that data is not accessible or gets lost in case of a database server node crashing.

3.2 Implementation

There exist a variety of different NoSQL database concepts and implementations: key-value stores, document stores and extensible record stores [3]. To choose the appropriate NoSQL database management system for INKIDU three key requirements were considered:

- **Extensibility:** The possibility to migrate the full data model into the NoSQL database has to be preserved. This means the database has to be capable of storing objects of higher complexity than histograms and answer sets.
- **Indexing:** Besides the unique ID of every record set a second additional field has to be indexed.

– **Reliability:** The database itself and the driver have to work well with PHP. Especially the driver has to be well-engineered for simple and reliable use.

The requirements to store more complex objects and indexing more than one field led to the decision to use document stores. In order to find the right document store fulfilling all of the above key requirements Apache CouchDB and MongoDB were investigated. CouchDB offers some drivers developed by community projects, but without support. MongoDB offers one well developed proprietary driver, which is known to work well with PHP. Therefore MongoDB was chosen for this project.

A point not linked to the key requirements but still notable is the presence of atomic operations in MongoDB. Atomic operations allow changes to individual values. For instance to increment a value the modifier $inc can be used in an update command, $push adds a value at the end of an array and $pull removes it. These updates occur *in place* and therefore do not need the overhead of a server return trip [3]. This is especially of advantage for the updating of histograms, where often only one single value has to be increased.

3.3 Performance Results

In order to measure the performance of the reconstructed system the fully relational and the hybrid implementation were compared against each other in two different scenarios: Firstly, the generation of the visualized histograms. This includes the querying of the histogram, the calculation of the average and the standard deviation and the rendering of a histogram image displayed to the user. Secondly, the export of all answer sets of an inquiry, which includes the querying of all answer data belonging to an inquiry and formatting it into a CSV file. These scenarios were chosen, because they are the most affected parts of the reconstruction. The test procedures were supplied with 100 different anonymized real-life inquiries with one to 200 questions. The inquiries itself had participants within the range from 10 to 1000. During the test scenarios the following two values were measured:

– **Query Timings:** The time needed to query the database itself.
– **Processing Timings:** The time the whole process needed to complete including the query time.

The reason to distinguish between querying and processing timings is that we also have to take into account the time consideration for the optimization of the computation processes outside the database. Table 3.3 shows the query timings and Table 3.3 shows the processing timings. The measurements show that the average querying time was reduced by about 30% to 95% and the average processing time by about 93% to 98%.

Histograms could have also been realized using a relational database. However all reasons of Section 2.2 to store answer sets in a NoSQL database are true for histogram data as well. These data are also semi-structured and change

in size dynamically. Therefore the overall performance improvements are the result of the combination of the relational database with a NoSQL database. Another possible approach - using an object database - was not investigated in this project.

	Histogram Querying (μs)		
	Min	*Avg*	*Max*
Relational	12	36	279
Hybrid	2	24 (**–30%**)	167
	Data Export Querying (μs)		
	Min	*Avg*	*Max*
Relational	14	4423	83699
Hybrid	3	200 (**–95%**)	8791

Table 2. Query timings for histogram and data export querying.

	Histogram Generation (μs)		
	Min	*Avg*	*Max*
Relational	179	166695	2355661
Hybrid	113	3983 (**–98%**)	116779
	Data Export (μs)		
	Min	*Avg*	*Max*
Relational	155	114837	2617227
Hybrid	141	8380 (**–93%**)	193134

Table 3. Processing timings for histogram generation and data export.

4 Conclusion

This paper illustrates an approach to distribute a fully relational data model into a relational and a non-relational part in order to deal with different requirements concerning consistency, availability and partition-tolerance. Before partitioning a relational data model into multiple parts, the choice of the database(s) has to be made carefully. Especially with regard to NoSQL databases, there is a large variety of different types of data stores available. In this project, the decision to select MongoDB as NoSQL database management system has been strongly influenced by the presence of a reliable driver for database connection and atomic operations, which provide the ability to make simple update operations as fast as possible.

The data model of the INKIDU inquiry application is separated in two parts: The structured part is still stored using the relational DBMS MySQL, which guarantees ACID transaction control. The semi-structured part, the answer sets of a questionnaire, is stored in the document store MongoDB. This reduces the amount of stored data by more than 50% because three of the four indexes can be omitted.

In a second step histograms are stored to increase the performance of a questionnaire's statistical analysis - a core feature of an inquiry application. MongoDB is used to store histogram data because they are semi-structured, too. The time the queries take to retrieve the histogram and export data has been reduced by about 30% and 95%. The overall performance for the processes of generating histograms and exporting data has increased by 98% and 93% respectively.

Acknowledgements

We gratefully acknowledge Wolfgang Ertel for collaboration and Michel Tokic for his valuable feedback on this paper.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From relations to semistructured data and XML. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
2. Brewer, E.A.: Towards robust distributed systems (abstract). In: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing. p. 7. PODC '00, ACM, New York, NY, USA (2000)
3. Cattell, R.: Scalable SQL and NoSQL data stores. SIGMOD Record 39(4), 12–27 (2010)
4. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. In: Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation. vol. 7, pp. 205–218 (2006)
5. Comer, D.: The Ubiquitous B-Tree. ACM Computing Surveys 11, 121–137 (1979)
6. Dean, J., Ghemawat, S., Inc, G.: MapReduce: Simplified data processing on large clusters. In: Proceedings of the 6th Symposium on Opearting Systems Design & Implementation. USENIX Association (2004)
7. Evans, E.: Eric Evans' Weblog. `http://blog.sym-link.com/2009/05/12/nosql_2009.html` (may 2009), retrieved: 2012-03-25
8. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California (2000)
9. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33, 51–59 (Jun 2002)
10. Leavitt, N.: Will NoSQL Databases Live Up to Their promise? Computer 43, 12–14 (February 2010)
11. Lerner, R.M.: At the Forge - Redis. Linux Journal 2010(197) (Sep 2010), `http://www.linuxjournal.com/article/10836` (Retrieved: 2012-02-27)

12. MongoDB: BSON. http://www.mongodb.org/display/DOCS/BSON, retrieved: 2012-02-26
13. Ramalho, L.: Implementing a Modern API for CDS/ISIS, a classic semistructured NoSQL Database. In: Todt, E. (ed.) Forum Internacional do Software Livre, XI Workshop sobre Software Livre, Porto Alegre. vol. 11, pp. 42–47 (2010)
14. Riva, C., Laitkorpi, M.: Designing Web-Based Mobile Services with REST. In: Di Nitto, E., Ripeanu, M. (eds.) Service-Oriented Computing - ICSOC 2007 Workshops, Lecture Notes in Computer Science, vol. 4907, pp. 439–450. Springer Berlin / Heidelberg (2009)
15. Robinson, D.: Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faster. Emereo Pty Ltd, London, UK, UK (2008)
16. Xiang, P., Hou, R., Zhou, Z.: Cache and consistency in NOSQL, vol. 6, pp. 117–120. IEEE (2010)