Application-Mimes
An Approach for Quantitative Comparison of SQL - and NoSQL-databases

Martin Kammerer and Jens Nimis
University of Applied Sciences Karlsruhe, Faculty of Management Science and Engineering,
Moltkestr. 30, 76133 Karlsruhe, Germany

Keywords: Information System Evaluation, Benchmarking, Database Systems, SQL, NoSQL.

Abstract: Due to the rise of NoSQL systems over the last years, the world of commercially applicable database systems has become much larger and heterogeneous than it was ever before. But the opportunities that are associated with the upcoming systems have also introduced a new decision problem that occurs in the information system design process. Once, benchmarking has helped to identify the proper database product among the de facto standard SQL systems. Nowadays, functional and non-functional properties of database systems and their implication on application development are so divergent that not all systems that come into account for realisation of a specific application can be covered by the same benchmark. In this paper we present an approach for experimental comparative information system evaluation that allows for well-grounded selection among diverging database systems. It is based on the concept of so-called application-mimes, i.e. functionally restricted implementations that focus exclusively on the information systems' interaction with data management and try to mimic the target systems behaviour in this respect as realistic as possible.

1 INTRODUCTION

The prime of modern, non-relational databases, aka. NoSQL databases (Redmond and Wilson, 2012; Mohan, 2013), and the rise of rethought relational databases, aka. NewSQL databases (Stonebraker, 2012), in its aftermath have lead to an enormous growth of information systems' design space. Resulting is a non-trivial decision problem on how to realize data management of future information systems with heavy influence on their system architecture and hence on their performance.

There are cases, where it seems to be easy to select the best-suited data management solution, as the indication by the information system to be built is rather extreme and distinct. If one tries to build a traditional single-tenant accounting system, it is reasonable to choose a SQL database for its strength and maturity in the OLTP area. For a new facebook NoSQL databases might be a more appropriate choice, as scalability is major concern. But which data management system fits best in not-so-clear application scenarios? Obviously, a systematic approach is needed to support the depicted decision problem.

Like to be shown in the second section, traditional database benchmarking merely regards aspects like throughput and response time, but nowadays there are a lot more dimensions in which the data management component candidates under examination for the respective information system can differ: functional properties (determined by data model, data access, etc.), and non-functional properties (consistency model, scalability, robustness, maintainability, TCO, etc.). The parameters covered by traditional benchmarks are most often found in the second category and – from a different point of view – could also be termed as qualities of service (QoS).

As an example, relational databases usually offer the very expressive query language SQL, which coins their pseudo-synonymic label. In contrast, NoSQL databases frequently offer reduced programming interfaces and leave more complex querying tasks to the application (layer). On the other hand, some of them include additional system components like Web-servers that in general are not subsumed in SQL databases.

The most considerable difference between SQL and NoSQL, however, is the consistency model.

DOI: 10.5220/0004896702560263
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 256-263
ISBN: 978-989-758-027-7
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

While SQL databases very often implement the ACID model (Haerder and Reuter, 1983) with its extensive isolation and recovery guarantees, many NoSQL systems restrict themselves to the BASE model (Terry et al., 1994) that reduce guarantees in favour of scalability.

As a consequence, when it comes to information system architecture and database selection decisions, it is not suitable to define a workload and run a traditional benchmark – sometimes it is not even possible, because available benchmarks are not general enough to measure all available, reasonable candidate data management systems. In this paper, we propose a more comprehensive approach based on the comparative experimental evaluation of data management systems using simplified prototypical information system implementations, which mime the characteristics of the later application. Our universal approach can be used for comparisons of any types of database systems: NoSQL vs. SQL, NoSQL family A vs. NoSQL family B or NoSQL system A vs. NoSQL system B both from the same family.

Outline. Section 2 gives an overview on the respective state of the art in database and information system evaluation and thereby also focusses the addressed problem. In Section 3 we propose the approach of comparative experimental information system evaluation. Our experiences with the approach result from four comparative studies between pairs SQL- and NoSQL-based information systems that we have conducted. We take one of these studies as comprehensive example to illustrate the approach in Section 4 and thereby also give guidance for potential adopters. The other studies are summarized in Section 5 for a critical discussion, while Section 6 concludes the paper and gives an outlook on on-going and forthcoming research activities.

2 RELATED WORK

There is a variety of existing traditional database benchmarks with different approaches, but none of them is applicable for an extensive comparison of SQL- and NoSQL-database systems. These benchmarks can roughly be divided into two benchmark types: Type one only measures basic database performance metrics, like throughput and/or response time. Benchmarks of type two simulate a comprehensive realistic use case and therefore have their specific metrics.

The most common database benchmarks are provided by TPC (Transaction Processing Performance Council, 2013). For example, the TPC-C benchmark (Transaction Processing Performance Council, 2013b) simulates a complete business activity, where simulated terminal operators execute transactions against a database. The measured metric is the number of New-Order transactions executed per minute. However, this benchmark per definition needs the respective tables, such that it is not applicable to most NoSQL-databases. On the other hand, like shown by Thanopoulou et al. (2012), the TPC-benchmarks are up- and down-scalable, and have a specific, well-defined workload, which results in an apples-to-apples comparison on real use cases.

With the rise of NoSQL-databases, new benchmarking concepts were needed. To close this gap Yahoo! Cloud Serving Benchmark (YCSB) (Cooper et al., 2010) has been introduced. The main focus of the YCSB benchmark is to provide a reasonable comparison between scalable databases, independent of specific data models or business use cases. To accomplish this, the benchmark consists of two tiers: Tier 1 - Performance and Tier 2 - Scaling. The workloads of the performance-tier are very simple and just execute heavy CRUD operations. There are no specific database management system requirements concerning the ability for JOIN queries. With the scalability-tier YCSB measures scaleup and elastic-speedup (DeWitt, 1993) behaviour of clustered database systems. They describe the performance impact of a database-cluster as the number of machines increase. The results of the YCSB benchmark can give a first advice for choosing a well-suited database.

However, experiences with complex information systems (Doppelhammer, et al., 1997) have shown that isolated benchmarking is not sufficient for elaborate database selection and current research on database evaluation primarily focuses on qualitative (Hecht and Jablonski, 2011) or conceptual comparisons (Bernstein and Das, 2013) in the NoSQL field.

3 PROPOSED APPROACH

In this section we present a hands-on approach for experimental comparative information system evaluation. The core of the approach is to create, test, and compare multiple simplified prototypical versions of the target information system – so-called application-mimes. The approach's primary focus is to qualify its users to conduct extensive and sound comparisons between different database systems within their later application scenario.

3.1 Developing Application-mimes

An application-mime is a piece of software, which imitates the typical behaviour of a target application scenario for evaluation purposes. Therefore it has the same functional and non-functional requirements as the scenario sets, but can have very differing realizations. In particular, we use multiple realizations of an application-mime – with specific focus on performance impact of data management – to allow for extensive comparisons between SQL- and NoSQL-database systems.

3.1.1 Specifying Functional and Non-functional Requirements

The first step for developing an application-mime is to specify the requirements of the target application scenario. Therefore it is necessary to convert its use cases into functional and non-functional requirements.

The functions can be described as UML Use Case Diagrams (Object Management Group, 2013), where it is important to get an idea of what is the main usage of the application and what processes are essential for serving its users.

On the other hand it is important to record explicitly the non-functional requirements of the application scenario. Examples could be consistent datasets, horizontal scalability, or security requirements, but also the use of a specific programming language or other issues of a more organizational nature.

Amongst all requirements, it is important to identify disqualifying criteria, as they are the major source for selection of candidate database systems that should be considered more closely in the evaluation process. Qualitative database comparisons like in Hecht and Jablonski (2011) can be used to facilitate this task.

The results of this first step are general and applicable to all realizations of application-mimes.

3.1.2 Focusing the Application-mime

In general an application-mime simulates specific application behaviours, but we want to use it for evaluating purposes on data management. Therefore we try to give it a focus on the database system. As a consequence, we first take a look on the different selected database systems, and for each of them try to map the requirements of the application-mime to their matching functional and non-functional abilities and properties.

Some common database functions, like joining tables, could be missing in NoSQL databases, resulting in a need for additional software routines in the application to fill this gap. This is one reason why comparing SQL- and NoSQL-databases on a basic CRUD-level is misleading.

On the other hand, one can take rather unexpected functions into consideration as some NoSQL-databases could have surprising helpful features compared to their SQL predecessors. When these features fit into the application-mime they should be used for a "fair" and useful comparison.

3.1.3 Integrating the Evaluation Layer

The evaluation layer is the part of the simplified prototypical system that is responsible for measuring the performance impact of the focused part of the application-mime. For its implementation, first, it is important to determine, which use cases of the target application scenario should be simulated. Derived from the result, it is clear, which processes have to be measured, and how a respective index can be calculated.

The second step is to decide how the use case simulation will be implemented. For this task, there exist a number of specific frameworks, which could be used, like JMeter from Apache Foundation (Erinle, 2013).

The evaluation layer implementation at the end has to provide a generally applicable tool, e.g. a set of parameterized JMeter instances, which should be reused in all realizations of the application-mime to allow for a reasonable comparison. It prepares and triggers the application scenario's events and measures and assesses the respective actions.

3.1.4 Completing the Application-Mime

The big picture of the complete application-mime should have started to become clear in the course of the database mapping and evaluation layer integration steps. However, some parts, which glue together data management and evaluation layer, are probably still missing.

The missing functions are bundled in the application-layer, which mimes the application-behaviour towards the focused part – in this case the database system. To identify the missing functions one has to consider, which functions are carried out by the database system (and which not) and how the evaluation system works and is interfaced. As a result the absent functions are identified and their respective building blocks complete the application-mime architecture.

As a consequence of the functional differences between the candidate database systems, it is inevitable to repeat this step for each of them.

3.1.5 Programming Application-mime Versions

The application layer is used by the evaluation layer, which tries to put the whole system under heavy load during test runs. Therefore, it is important to achieve good horizontal scalability of the application layer, as there could be multiple isolated application layer instances, which are jointly using the database system.

Another point to concern is best-effort implementation of all functions in the application layer independent of the respective candidate system. The differing built-in functionality of the database systems and the interdependence between their functional and non-functional properties can lead to a complete corruption of the comparisons' results, if the application layer is not implemented optimally with respect to the candidate system. As a consequence, all available features must be fully exploited and used in the intended fashion.

Figure 1 shows the application-mime architecture, which results from the steps above.

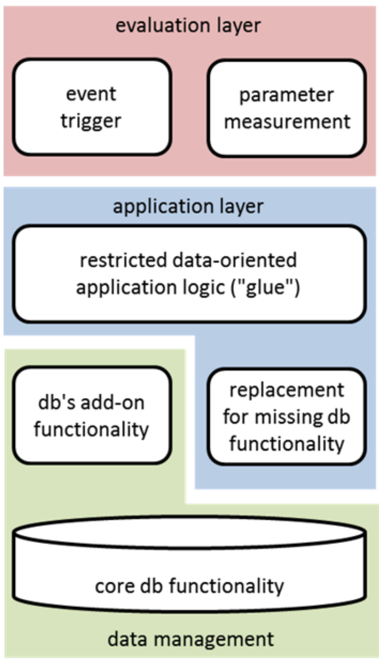
evaluation layer
event trigger
parameter measurement
application layer
restricted data-oriented application logic ("glue")
db's add-on functionality
replacement for missing db functionality
core db functionality
data management

Figure 1: Application-mime architecture.

3.2 Testbed Setup

Testbed setup is strongly influenced by the planned deployment environment of the application scenario. To make the results of the comparison as realistic as possible it would be optimal to use the later deployment environment itself – or at least to use a comparable testbed, for which similar financial and/or technical restrictions apply.

If this is not realistic, it is a minimum requirement to run the tests for each application-mime realization on identical or at least on similar hardware. It has to be noted that similarity in this case could also be measured in financial terms as, e.g. in some application scenarios relational databases preform best on one centralized machine while NoSQL has its best results on a decentralized cluster of equal costs. This consideration is not only true for database system and application layer but also for evaluation layer to assure that the generated workload is identical for all test runs.

As the testbed most probably is a distributed system, also network and other infrastructure components must be chosen deliberately. Otherwise, bottlenecks especially between evaluation and application layer could occur and lead to distorted results. One way to prevent these bottlenecks is to build test servers, which integrate evaluation layer and parts of the application layer on the same node. This reduces network traffic between them and can increase the load on the focused evaluation part.

From our experience, we advise not to underestimate the requirements of a significant evaluation layer. Modern database systems are extremely powerful, and it needs a fair amount of hardware on evaluation layer side to push them to their limits, i.e. into the area of interesting and relevant evaluation results.

3.3 Workload Design and Execution

The workload to be executed by the application-mimes during evaluation is derived from the application scenario's use cases and thereby is affected in a number of properties.

For example, the database systems under evaluation need an initial content against which the operations of the mimes' application layers can be conducted. This content needs sufficient volume and complexity to provide a challenge for the database systems with respect to the planned number and frequency of requests issued by the processes of the evaluation layer.

The number of simulation processes itself is determined by the amount of available hardware in total and by the ratio it is dedicated to the mime's database, evaluation and application layer. As the impact of each process on the application mime's performance can be varying for different use cases and parameterizations of the application scenario, finding the proper ratio between evaluation and application layer hardware can be a difficult task.

The results of the operations and also their QoS-parameters are measured within the evaluation layer, which as a consequence needs the capability to log and export them in an interpretable way. It is recommended to use external generic tools for analysis and visualisation of results to keep the application-mimes as lean as possible and reduce development efforts.

4 ILLUSTRATION BY EXAMPLE

To illustrate the approach introduced in the last section, we give an example for its use in a specific application scenario, i.e. a project relationship management system.

4.1 Application-Mime

We imagine that we would have to build such a system for a mid-size company. The system should be able to manage projects and their associated resources, especially project staff, from project acquisition over execution to its accomplishment. Most phases of the project follow predefined or project-specific processes and occurring data are manifold and closely interrelated, ranging from address records and status messages to documents like contracts or bills.

As our company has a large number of simultaneously running projects and their staffs have to work with the system on a day-by-day basis, the system must have low latency for good user acceptance. Hence, we have to develop an efficient information system and make appropriate well-informed choices for the system's components.

4.1.1 Requirements

The depicted application scenario implies a number of use cases with relevance to data management. The system has to provide a comprehensive view on the company's projects from different angles: project staff needs a personal perspective to identify project status and upcoming tasks, project management needs an overview with insights into led projects, and upper management requires an enterprise perspective with aggregate information over all projects.

Examined in more detail, project management must be enabled to start new projects, define specific processes and their related (digital or personal) resources, identify and assign free resources, track progress and costs, and close projects after their completion. Staff and project management should be able to define personal profiles with individual competencies and experiences as a basis for staffing of new projects. Figure 2 shows the project relationship management system's functionality as UML Use Case Diagram.

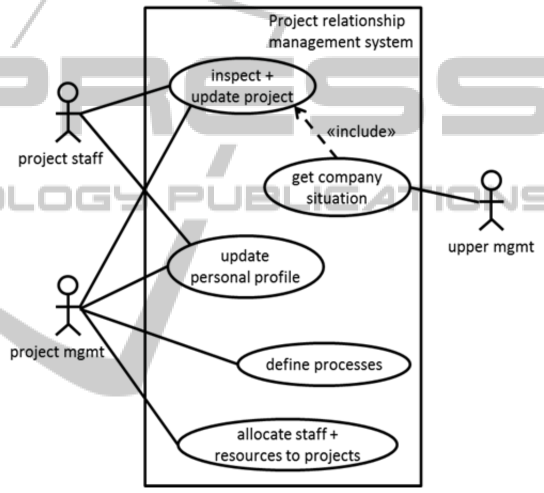

Figure 2: Functional specification of application scenario.

On a lower level, these use cases lead to a number of functional requirements with influence on data management. Persistent entities for projects, processes, employees and information objects with the respective properties have to be provided as well as numerous types of relationships between them that can establish and change over time. In general, we estimate that read operations appear up to ten times more often than write operations.

Along with functional requirements, also non-functional requirements can be derived from the use cases. Most important, the project management system has to be regarded mission critical, and thus must have high availability. This can be supported by redundancy through distributed replication. Concurrent access by different users accounts for a multiuser client/server approach. As allocation of resources is an important functionality and double bookings have to be avoided, transactional consistency on data level would facilitate application programming.

Our target system only manages project meta-data, i.e. not the created project results, and is restricted to internal company users. Thus, data volume and number of users are moderately high, especially as compared to modern public web applications that gave birth to NoSQL data management. Without loss of generality, to follow company standards, Java is chosen as programming language for client as well as server.

As no disqualifying criteria can be identified among these requirements, the choice of candidates is not restricted to certain database systems or families.

4.1.2 Focus

The main goal of experimental comparative information system evaluation is the identification of the best suited data management system for a specific application scenario. Thus, we have to identify reasonable candidates (or candidate families) to compete with each other.

As mentioned before, the application scenario under investigation has heterogeneous data interrelated closely in many different ways. It should be possible to access data by navigation and also by searching via descriptive expressions. The need for transactional consistency under distribution and the read/write-ratio assumed (10:1) will also influence database performance.

The first candidate we select for evaluation is MySQL (Schwartz et al., 2012) as it is a mature and universal relational database that provides powerful descriptive querying capabilities. It is also subject to a number of similar examinations in related papers. Second candidate is the Neo4J NoSQL-database (Robinson et al. 2013) as it is popular amongst the family of graph database systems. Such a database system naturally matches the network character of data and allows for easy navigation access. Based on personal preferences any other candidates of the same database types could qualify.

To restrict evaluation effort we concentrate on these two, quite disparate, candidates, and have to build the two respective application-mimes. In case of doubt on the selection of the database candidates, or if it would have shown during evaluation that none of the candidates suits the requirements in acceptable quality, the comparison easily could have been extended to additional database systems and their mimes, following the above pattern.

4.1.3 Evaluation Layer

The evaluation layer is responsible for driving the experiments. In the example application-mimes graphical user interfaces and user interactions are replaced by Apache JMeter instances that also record the results of the triggered operations. JMeter itself makes use of application layer functions that do not implement the fully-fledged application logic but provide a restricted logic that makes reasonable database calls according to the respective application programming interface. The application layer functions also unify the database calls' results into an easily processable form for the (virtual) real application logic.

4.1.4 Completion

An example functionality that must be realized during completion of the MySQL-application-mime is support for navigation data access. Navigation in the relational version of our project relationship management system means joining tables. Unfortunately, joins over multiple distributed tables within a MySQL-cluster are a very inefficient operation, which could result in a decisive drawback during comparison, if it is used in a naïve way.

However, for a fair comparison it is mandatory to use all candidate databases in the best way possible. For the case of MySQL and distributed joins this means that the respective tables to join have to be read completely and un-joined by application function's database calls and matching of keys and foreign-keys happens in the application logic itself. "Fair" implementations of application-based joining use read operations on tables pre-ordered by their keys and foreign keys respectively to increase processing- and memory-efficiency.

4.1.5 Programming Versions

Besides making best-effort use of the candidate database systems' capabilities as described above, choosing the proper programming interface is another important concern. In the case of the MySQL-application-mime we use the standard java-mysql-connector and the Neo4J-cluster is accessed via its REST-API. An additional performance optimization for Neo4J is to send write operations directly to the master node of the cluster and to distribute read operations equally amongst all nodes (i.e. master and slaves).

4.2 Testbed

As testbed for both application-mimes a cluster existing of 16 simply equipped HP Microservers has been used, i.e. MySQL and Neo4J both have been set up as clusters with their according dedicated management nodes (MySQL Config-Server and Master-Node resp.). A considerable portion of the testbed's hardware has been used for JMeter instances in order to avoid that they become the system's bottleneck with significant influence on the comparison's results.

To underline or refute the comparison's findings, it could be worthwhile to execute the MySQL-application-mime on a centralized system, as, like already mentioned, some operations degrade in distributed settings. For a fair comparison between centralized MySQL- and decentralized Neo4J-results the original costs of both hardware variants should be the same (i.e. in this case roughly 5000 EUR in early 2013).

4.3 Execution

For each application-mime a large number of test runs have been conducted and their results have been compared pairwise and in aggregation. Input parameters under variation have been derived from the application scenario, like e.g. number of projects, number of members/documents per project, number of concurrent users, etc..

The main output parameters that have been recorded and analysed are throughput and response time. As this paper is focussed on the comparison method and not on the specific comparison itself, the results are not discussed in detail at this point. However, for the specific application scenario, it has shown that the graph database outperformed the relational database for all parameter settings. This can be attributed to the high interrelation degree of the project data.

5 CRITICAL DISCUSSION

The presented comparison method has been developed in a research project running over the past two years, which had the goal to provide a guideline for reasonable selection of data management solutions for a given application scenario. Special attention was given to application scenarios where both, SQL- and NoSQL-databases, qualify as data management component. During the course of the project we have conducted a number of SQL vs. NoSQL comparisons, where another three have followed the above approach as traditional benchmarks were not appropriate for the reasons described in Section 2.

The first of the additional application scenarios is a social analytics scenario where data from different social media sources are gathered and analysed, e.g. for marketing purposes. Like in all other scenarios, MySQL has been chosen as relational database, and as aggregation operations are of major importance in data analysis, HBase Wide Column Store was selected as its competitor. (See Redmond and Wilson, 2012, for HBase and all other NoSQL-databases mentioned in this section.)

In the second scenario, a system for intra-organisational product tracking should be investigated. Like it is common in modern manufactories, products and precursors are observed by means of labels (e.g. RFID) and sensors. From a data management point-of-view, the scenario is characterized by a very low read/write-ratio, i.e. that position actualisations occur far more often than user searches. This time, the REDIS key/value-store was the NoSQL-variant to compete against MySQL.

As last comparison an online product data catalogue was the scenario under inspection. The target system should enable manufacturers to publish product specifications, descriptions and manuals online and consumers to rate and comment on the products. As all product-related information in this scenario can be interpreted as linked semi-structured objects, the MongoDB document store seemed as reasonable alternative to MySQL.

For all three application scenarios the respective MySQL- and NoSQL-application-mimes were built, i.e. six mimes in total, and compared pairwise. All comparisons led to significant and interpretable results, were in one case the SQL-variant was preferable for most parameter settings and in the other cases, the NoSQL-databases were superior. We believe that these results are strong evidence for successful selection of data management components in the discussed scenarios and currently do not see obstacles for transfer to other scenarios.

However, it is undisputable that implementation of application-mimes for experimental comparison introduce a substantial effort into information systems' component selection and architectural design process. This investment is partially reimbursed, when it comes to system's implementation, as this step will benefit from the experiences made during application-mime development.

6 CONCLUSIONS AND FUTURE WORK

In this paper we have presented an approach for experimental comparative information system evaluation based on application-mimes that restrict the target systems application functionality to database-relevant operations. This allows for reasonable and well-grounded selection among modern database systems, where traditional benchmarks fail due to the systems' and applications heterogeneity. The price for this support is the necessity to build a specific application-mime for each target application scenario and candidate database system to be evaluated, and the costs to build such a mime depend on the applications extent and complexity.

In future, we will investigate on two aspects to advance our approach. First we would like to offer a better support for the non-trivial step of candidate selection. Currently, this is only supported by the identification of disqualifying criteria among the application scenarios requirements. As shown in our example, many candidates can qualify, if no such criteria are identifiable. A positive list to propose candidates based on application scenarios' requirements and database system properties would be preferable.

The extension of this idea is also the motivation for the second question to work on: how could experiences from the experiments be distilled into a more abstract conceptual approach, which allows for decision making based on matching application requirements to database system properties without or with less specific experimentation.

ACKNOWLEDGEMENTS

The work presented in this paper has been partially funded as project within the Karl-Steinbuch research program of the Baden-Württemberg foundation coordinated by mfg Medien- und Filmgesellschaft, Stuttgart, Germany.

REFERENCES

Bernstein, Ph. A., Das, S., 2013. Rethinking eventual consistency. In *SIGMOD '13, ACM SIGMOD International Conference on Management of Data.* ACM.

Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R., et al, 2010. Benchmarking cloud serving systems with YCSB. In: *SoCC'10, 1st ACM symposium on Cloud computing*, ACM.

DeWitt, D.J., 1993. The Wisconsin Benchmark: Past, present and future. In J. Gray, editor, *The Benchmark Handbook*. Morgan Kaufmann.

Doppelhammer, J., Höppler, Th., Kemper, A., Kossmann. D., 1997. Database performance in the real world: TPC-D and SAP R/3. In *SIGMOD '97,* ACM SIGMOD international conference on Management of data, ACM.

Erinle, B., 2013. *Performance Testing with JMeter 2.9*, Packt Publishing, Birmingham

Haerder, Th., Reuter, A., 1983. Principles of transaction-oriented database recovery. *ACM Comput. Surv. 15, 4,* ACM.

Hecht, R., Jablonski, S., 2011. NoSQL evaluation: A use case oriented survey. In *CSC'11, Int'l Conference on Cloud and Service Computing,* IEEE.

Mohan, C., 2013. History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla. In *EDBT '13, 16th International Conference on Extending Database Technology*. ACM.

Object Management Group, 2013. UML Resource Page http://www.uml.org (last visited: 2013-09-17)

Redmond, E., Wilson, J.R., 2012. *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*. Pragmatic Bookshelf, Dallas.

Robinson, I., Webber, J., Eifrem, E., 2013. *Graph Databases*. O'Reilly, Sebastopol.

Schwartz, B., Zaitsev, P., Tkachenko, V., 2012. *MySQL High Performance*. O'Reilly, Sebastopol.

Stonebraker, M., 2012. New opportunities for New SQL. Commun. ACM *55, 11.* ACM.

Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M.J., Theimer, M.M., Welch, B.B., 1994. Session guarantees for weakly consistent replicated data. In *PDIS'94, 3rd Int'l Conference on Parallel and Distributed Information Systems*, IEEE.

Thanopoulou, A., Carreira, P., Galhardas, H., 2012. Benchmarking with TPC-H on Off-the-Shelf Hardware: An Experiments Report. In *ICEIS'2012, 14th Int'l Conference on Enterprise Information Systems*, SCITEPRESS.

Transaction Processing Performance Council, 2013. *About TPC*. http://www.tpc.org/information/about/abouttpc.asp (last visited: 2013-09-16)

Transaction Processing Performance Council, 2013b. *Introduction to TPC-C*. http://www.tpc.org/tpcc/default.asp (last visited: 2013-09-16)