History Repeats Itself: Sensible and NonsenSQL Aspects of the NoSQL Hoopla

C. Mohan
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA
+1 408 927 1733
cmohan@us.ibm.com

ABSTRACT

In this paper, I describe some of the recent developments in the database management area, in particular the NoSQL phenomenon and the hoopla associated with it. The goal of the paper is not to do an exhaustive survey of NoSQL systems. The aim is to do a broad brush analysis of what these developments mean - the good and the bad aspects! Based on my more than three decades of database systems work in the research and product arenas, I will outline what are many of the pitfalls to avoid since there is currently a mad rush to develop and adopt a plethora of NoSQL systems in a segment of the IT population, including the research community. In rushing to develop these systems to overcome some of the shortcomings of the relational systems, many good principles of the latter, which go beyond the relational model and the SQL language, have been left by the wayside. Now many of the features that were initially discarded as unnecessary in the NoSQL systems are being brought in, but unfortunately in ad hoc ways. Hopefully, the lessons learnt over three decades with relational and other systems would not go to waste and we wouldn't let history repeat itself with respect to simple minded approaches leading to enormous pain later on for developers as well as users of the NoSQL systems!

Caveat: What I express in this paper are my personal opinions and they do not necessarily reflect the opinions of my employer.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems – *distributed databases, query processing, relational databases, transaction processing*

General Terms

Algorithms, Design, Documentation, Languages, Management, Measurement, Performance, Reliability, Standardization

Keywords

APIs, Data Models, DBMS, HBase, Hype, In-memory, JSON, MongoDB, NoSQL, Optimization, RDBMS, SQL

EDBT/ICDT '13, March 18–22, 2013, Genoa, Italy

1. INTRODUCTION

Over the last few years, many new types of database management systems (DBMSs) have emerged which are being labeled as NoSQL systems. These systems have different design points compared to the relational DBMSs (RDBMSs), like DB2 and Oracle, which have now existed for about 3 decades with SQL as their query language. The NoSQL wave was initially triggered not by the traditional software product companies but by the non-traditional, namely the Web 2.0, companies like Amazon, Facebook, Google and Yahoo. Some of the NoSQL systems which have emerged from such companies are BigTable, Cassandra [14], Dynamo and HBase. Different types of NoSQL systems have emerged. Some of the types are: key-value stores, document DBMSs, graph DBMSs and column-oriented stores. There are older non-relational systems that are also being classified these days as NoSQL systems: object-oriented (OODBMSs) and XML DBMSs.

Of late, a good amount of the momentum for the NoSQL developments is also being provided by a number of smaller software companies which are fuelled by venture funding [22] and/or the open-source movement. Some of the systems emerging from such efforts are Aerospike (formerly Citrusleaf), Couchbase, MongoDB and Riak. Oracle has also enhanced its previously acquired product Berkeley DB Java Edition with additional features that are found in many NoSQL products. This product is called Oracle NoSQL Database [11]. For XML data and graph data, IBM supports them via its enhancements to DB2 (pureXML and SPARQL NoSQL Graph Store features). Also, HBase is included as a component of the IBM InfoSphere BigInsights product which is targeted at the big data market.

The goal of the paper is not to do an exhaustive survey of NoSQL systems (see [10, 12, 13, 18, 21, 22] for more information on them), nor is it to get into too many details of any specific systems. The aim is to do a broad brush analysis of what these NoSQL developments mean - the good (sensible) and the bad (what I have termed in the title of the paper, in a tongue-in-cheek way, as NonsenSQL) aspects! This paper is a follow up to my year-ago series of blog posts focused on NoSQL systems [5]. Since that time, in spite of all the hype about NoSQL systems, it has been good to see thoughtful articles like [20] which have tried to provide some reality checks and bring about some sanity to the discussions!

The emergence of the NoSQL systems has been triggered by a number of considerations in the context of certain types of applications, typically Web 2.0 ones, for which RDBMSs were found to be inadequate for a number of reasons:

- The limitations of the relational model for modeling the Web 2.0 kind of data were found to be the most problematic. Web data like logs of activities in an e-commerce site or data managed by social media applications like Facebook is not very structured and, even when it is structured, the structure changes a lot over time. For such data, the relational model is found to be very rigid. Traditionally, RDBMSs haven't been very good in their support for schema evolution. More recent enhancements to RDBMSs to support native storage of XML and querying via XQuery aren't considered as providing sufficient flexibility by some people for their needs.

- Compared to when RDBMSs were originally developed three decades ago, with the emergence of popular new programming languages like Java, JavaScript, Perl and Python, native support for data interchange or data storage formats like JSON (JavaScript Object Notation) and BSON (Binary JSON) were considered to be essential or at least desirable in the modern DBMSs.

- Another perceived limitation of RDBMSs was the need for the programmers writing data manipulation code having to learn SQL for that purpose in addition to being experts in a regular programming language in which the SQL calls would be embedded. This is a programmer productivity argument that favors a single language for both imperative programming and persistent data manipulation. This is in many ways a revival of the topic of persistent object systems (POS) which have been worked on since the 1980s with many conferences/workshops devoted to the topic! In fact, [9] discusses in detail how concepts in the MUMPS language from the 1960s have been used to produce two products, GT.M (first deployed in 1986!) and InterSystems Caché, which are presented as superior NoSQL products!

- In the minds of some people, access to the source code of the DBMS being used and the freedom to be able to modify that code to suit their specific needs were also relevant factors. This encouraged the adoption of the open source paradigm in the case of a number of NoSQL systems. For some users, such open source software is attractive from a cost viewpoint even if they have to pay for the services provided by a software company that supports such a product. Companies like 10Gen, Basho, Couchbase and DataStax are providing commercial support for the open source NoSQL systems MongoDB, Riak, Couchbase and Cassandra.

- For certain types of application requirements like very large volumes of data to be handled and the response times that needed to be provided for some simple data accesses, RDMSs' pathlengths and other costs were considered to be unacceptable. This has caused a number of systems to adopt various types of in-memory and data partitioning ("sharding") technologies and efficient access methods like hashing for single data item accesses based on a primary key.

- With the emergence of cloud computing and huge data centers being set up by the large Web 2.0 companies for their own usage and sometimes for serving the compute and storage needs of others via cloud services, the desire to use commodity servers/disks has become a dominant theme. This in conjunction with the need to handle much larger volumes of heterogeneous, semi/unstructured data has, in the minds of many people, warranted the development of much more scalable databases compared to what RDBMSs have traditionally supported.

- In conjunction with the desire to support databases scaling to hundreds, if not thousands, of servers, arose the consequent need to support more graceful ways of dealing with failures of individual nodes via sophisticated replication schemes.

- The consistency requirements of the new types of applications were also deemed to be less stringent than the ones demanded by traditional OLTP style applications for which the relational and non-relational transaction processing systems of the past were designed.

I am not fundamentally against the emergence of NoSQL systems or anything like that. I am glad that certain database requirements that had long been ignored or not handled as effectively by RDBMSs have been focused on by the NoSQL system builders. My concern has much more to do with how the latter are being architected and how the design choices being made are documented and rationalized.

Not all of the concepts/ideas currently hyped about in the context of NoSQL systems are that novel as their proponents would have you believe. For some of the NoSQL people either their memory is bad or they are too young to know personally from experience about what has been done in the past and the lessons learnt as different approaches were tried out. As I will discuss in detail later, many of the supposedly-novel ideas have been around for a long time.

In certain segments of the information technology (IT) and venture capital (VC) communities, there is too much hoopla associated with the NoSQL movement and some of that has also impacted the traditional database research community in industry and academia. At times there is an "anything goes" attitude amongst a significant fraction of the people using and/or working on such systems. Of late, it has become fashionable to discredit RDBMSs, and a significant chunk of the technologies that have been laboriously thought about and worked out over the last few decades. Some inconvenient/inadequate features of RDBMSs in certain contexts have been used as arguments to "throw the baby with the bath water" while coming up with alternatives. As some of us anticipated, many features which were initially considered unnecessary/undesirable are now being retrofitted to the NoSQL systems, in many cases in ad hoc and simple-minded ways which could lead to problems, if not now, in due course of time.

Having worked in the database field for more than 3 decades with a fair amount of impact on the research and commercial sides of this field (see http://bit.ly/cmohan), it pains me to see the casual way in which some recent system designs have been done and some supposedly new ideas get proposed/implemented. Not enough efforts are being made to relate these proposals to what has been done in the past and benefit from the lessons learnt in the context of RDBMSs. Not everything needs to be done differently just because it is supposedly a very different world now!

In the rest of the paper, I will delve into various specific attributes of the NoSQL systems and discuss where there are similarities to systems designed decades ago and where legitimately new approaches might be needed. The goal is to avoid the pitfalls with starting out with simple designs and then, after realizing their inadequacies, adding layers of patches to attempt to provide advanced functionality. This is a recipe for bad architectures and

consequently problems in the long term. It would be unfortunate if we don't learn from the mistakes of the past and let history repeat itself.

2. SOME HISTORICAL BACKGROUND

For the skeptical reader, it might be worth pointing out a few things. I am not an RDBMS bigot! I am open-minded about new ways of addressing problems and their solutions. I have invented and transferred technologies to not only IBM and non-IBM RDBMS products and prototypes but also to non-relational systems like MQSeries messaging system, Lotus Notes groupware/document system, WebSphere Application Server, FlowMark workflow management system, and Parallel Sysplex Coupling Facility in the mainframe and non-mainframe environments [4].

I am not trying to claim that I know everything about our industry/technologies or what matters when, or that I have definitive ideas about what the right evolutionary path for DBMSs is. I am merely trying to temper some of the marketing and technical hype associated with NoSQL and related areas, and to pass on some caveats and warnings based on my more than 3 decades of experience in the data management field. I am a very details oriented database person who has worked mostly on technologies relating to the core of different systems which manage persistent data of different kinds in distributed and clustered environments. In my writings and while designing my algorithms, I have tried hard to dig into what has been done in the past and document as much of my learning about the prior art and related work in my papers, crediting the people who did the prior work.

My comments aren't targeted merely at one NoSQL system or one set of people. I would like all sorts of people to give some attention to what I have to convey regarding NoSQL systems: entrepreneurs, end users, IT management, systems architects, designers, marketers, students, industrial researchers, academicians (pure and those who moonlight on the side as entrepreneurs and consultants), established little/big industry people, …

I have closely observed or taken part in the evolution of many systems the designers of which initially designed their systems thinking in a simple way but later on had to add more sophisticated functionality which they found out was very hard to do. Examples are IBM System/38's database functionality which was embedded in the horizontal and vertical microcode of the system, Lotus Notes which from its beginnings in 1989 has been architected in many ways like the NoSQL systems of today, and RDBMSs like mainframe version of DB2, Sybase and SQLServer, and OODBMs like ObjectStore which started out with page level locking as the smallest granularity of locking.

S/38 was built from the ground up as an object oriented system with a single level store concept. It had relational data management concepts embedded in the guts of the machine even though it didn't support SQL for a long time. It relied on the virtual memory paging subsystem and the file system for accessing and caching data in memory. There was no buffer manager as in other RDBMSs of that era. The granularity of latching during a call to the data manager was an entire table (locking was at record level). As the S/38 machines became more powerful and SMPs came into existence, latch conflicts became severe and the myriad things that took advantage of the table level latch became very painful to deal with.

Until R5, Lotus Notes had very ad hoc ways of handling recovery, no proper notion of transactions (as is the case with many NoSQL systems now) and many non-scalable internal design elements. Changing that system and adding log-based recovery and transaction semantics in R5 was quite painful [5]. Initially, it also didn't have a buffer manager but instead relied on the file system doing its caching and it used a single file for storing the whole database which consisted of a mish-mash of data structures of varying lengths.

Reducing the smallest granularity of locking from page size to something smaller was quite painful in RDBMSs/OODBMSs like DB2, Sybase, SQLServer and ObjectStore. The original lock granularity had been taken advantage of in many places in unobvious/subtle ways and those were very tricky to identify and fix [8].

Transforming a system like DB2, which was originally designed for a single SMP, to support a multi-system clustered environment with shared disks was a non-trivial piece of work that required major changes to the buffer/lock/log/recovery managers [4].

3. General Observations

I am really concerned about some of the design choices already made or being made in the case of some NoSQL systems. As they mature and what were initially considered as unnecessary features start creeping in (due to the slippery slope that these systems are on when they downsized significantly from the feature set of RDBMSs), they are going to suffer a lot with growing pains along the lines discussed for other systems in the last section. I am unsure of the extent to which the designers of such systems are conscious of these sorts of consequences of what they have chosen to do initially, in many cases in simple minded ways.

I tried to demonstrate in our original ARIES paper [8] the benefits to be had and the need for concurrently thinking about storage management, locking and recovery, unlike some layered approaches advocated in some earlier work. I also discussed numerous approaches to locking and recovery implemented in relational and non-relational systems which would be worth paying attention to as NoSQL systems evolve or new ones are built.

While there is a lot of talk by the NoSQL people about scalability, elasticity, etc., such design criteria seem to be applied in a spotty way in the design of their systems. For example, even systems which support incremental updates, as opposed to batch updates, don't seem to think of having to scale along the concurrency dimension by supporting finer granularity of locking/latching.

Way too much burden is being placed on the laps of the application writers or database administrators since even statement level atomicity isn't guaranteed when a single statement which updates more than one object encounters a failure of some sort or the other. Of course, this is a non-issue for many systems since only a few NoSQL systems support the functionality of being able to update multiple objects in a single statement!

The lack of standards due to most NoSQL systems creating their own APIs for data access and manipulation is also going to be a nightmare in due course of time with respect to porting applications from one NoSQL system to another. Whether it is an

open source system or a proprietary one, users will feel locked in. All the decades of evangelization that went on about the goodness of standards seems to have been forgotten in the context of NoSQL systems. Similarly forgotten are the benefits of high level query languages and application independence from access path considerations with the DBMS's optimizer worrying about such matters. Most NoSQL systems don't support high level query languages with built-in query optimization. Instead, they expect the application writer to worry about optimizing the execution of their data access calls with joins or similar operations having to be implemented in the application. Are application writers now going to try to master all the decades of sophisticated optimization technologies that have gone into query optimizers and then choose to implement a subset of them outside of the DBMS with all the restrictions that it entails? Even when some restricted SQL like query functionality is provided and some optimizations are also done by a NoSQL system, the optimizations are nowhere as sophisticated as those found in mature RDBMSs. With NoSQL systems supposedly being intended for managing vast amounts of data, simple minded optimizations would put us back to the early days of not-so-sophisticated RDBMS optimizers. We will have a repetition of history!

4. INDEXING

Most key-value stores have relied on hashing as the indexing mechanism which in turn has meant that they support only single object operations and hence a single node of the cluster being affected by a single API call. As they try to go beyond such a limited functionality and support general indexing, they are finding it that much harder to avoid having to access multiple nodes in processing a single API call. They are rediscovering the primary and secondary index concepts that have been supported for a long time in systems like Tandem's NonStop SQL and even pre-relational systems like IBM's IMS! Also, in the case of partitioned tables in DB2, features like local and global indexes have been implemented to provide partition independence, whether or not the system is operating in a shared nothing fashion [1]. These ideas are also being reinvented without reference to the past work!

Even notions like delayed updating of indexes that some people are reinventing now have been done in systems like Lotus Notes for two decades. Of course, some research literature has also explored such ideas even if not all of it was fully designed or implemented.

Index locking and recovery are also so much easer when high concurrency isn't a goal! So some systems have chosen to do some simple minded stuff and they will face hurdles in scaling those initial approaches to higher concurrency situations since the initial designs won't make that job easy.

5. DATA MODELS

While simplicity was touted by some as the reason for going with NoSQL compared to relational as the data model with SQL as the query language, in fact some of the NoSQL systems' data models are pretty complicated. Unlike in the case RDBMSs, for which a whole range of database design tools have been developed over the decades to make the database administrator's job easier for making design choices, as far as I know no such tools are available for the NoSQL systems with complex data models. Development of such design tools would be good and perhaps that is an area that the research community could focus on.

With widely varying data modeling constructs provided by the different advanced NoSQL systems, migrating from one such system to another would also be a nightmare. The tremendous efforts that went into standardization with SQL and even XQuery with the intention of making system migrations a lot easier seem to have been forgotten by the NoSQL people. While the big Web 2.0 companies might have been able to get away with their specialized software and the associated issues due to the sophisticated internal developers available to them, other enterprises, big or small, might not have the same luxury and hence they need to be much more cautious about such considerations in choosing their NoSQL systems for adoption.

While a company like Netflix took the step of migrating from its own data center to a public cloud and in the process also chose to migrate from an RDBMS to a NoSQL store [16], it is not clear how many traditional organizations would be able to do such a major transformation of their IT infrastructure that would also require significant application changes and some fundamental architectural changes. See [17] for the description of what all problems and choices Netflix had to wrestle with for migrating its applications.

6. DOCUMENT STORES

Document stores aren't really new as some people would like you to believe. Lotus Notes was first released in 1989. From the beginning, it was designed as a document store with many associated features like document level authorization, versioning, indexing, workflow and distribution/replication. It has even had field level authorization and encryption for security. It was initially designed for use in small workgroup environments and so the designers took some short cuts in terms of its various features which became a big problem later on when scaling had to be improved dramatically. Having suffered through those painful, internal design transformations to add those scalability features, it is troubling to see some of the NoSQL designers repeating those sorts of mistakes. For example, MongoDB was built with a single lock covering all the data in a node for concurrency control and the system doesn't have its own buffer manager but instead relies on the file system for caching. It also uses a single writer process. Couchbase, another document store, does replication of the whole document even if only a small part of it is changed during an update operation. While this might be acceptable for small documents, when documents are really large, this simple approach could have major performance consequences. Traditional content management systems have many sophisticated features which are missing in these modern-day document stores.

7. MYTHS ABOUT TRANSACTIONS

An often referred to distinguishing characteristic of NoSQL systems compared to traditional DBMSs is that the former don't care for ACID transaction functionality. In other words, NoSQL systems don't need to support the transaction concept. This is an oversimplification to say the least. As long as a NoSQL system supports incremental updates by concurrent set of users (as opposed to only single-threaded bulk or batch updates), even if multi-API-calls transactions aren't supported, at least within the internals of such a system some notion of transaction is essential to retain a certain level of sanity of the internal design and to keep things consistent. This is even more important if the system

supports replication and/or the updating of multiple data structures within the system even in a single API call (e.g., if there are multiple access paths which have to be updated). Similar points apply to locking and recovery semantics and functionality.

The above sorts of issues are real and were quite tricky to handle in Lotus Notes, which used very ad hoc ways of dealing with the associated complications, until log-based recovery and transaction support were added in R5 [6]. From its first release in 1989, Notes has supported replication and disconnected operations with the consequent issues of potentially conflicting parallel updates having to be dealt with. Even RDBMSs were late in dealing with that kind of functionality.

Even if at the individual object level, high concurrency isn't important given the nature of a NoSQL application, it might still be important from the viewpoint of the internal data structures of the NoSQL system to support high concurrency or fine granularity locking/latching (e.g., for dealing with concurrent accesses to the space management related data structures [7]).

Vague discussions about NoSQL systems and ACID semantics make many people think that RDBMSs enforce strong ACID semantics all the time. This is completely wrong if by that people imply serializability as the correctness property for handling concurrent execution of transactions. Even from the very beginning, RDBMSs (e.g., the original IBM Research relational prototype System R and products that came from it) have supported different degrees of isolation, in some cases even the option of being able to read uncommitted data, and different granularities of locking [2]. Even with respect to durability, in-memory RDBMSs like TimeTen and SolidDB which came much later, allowed soft commits, etc., trading off durability guarantees for improved performance. Even the age-old Airline Control Program (ACP, now called Transaction Processing Facility (TPF)), which powers the bulk of the world's airline reservation systems, made such tradeoffs.

This whole space of data management is a tricky business. The devil is in the details and it isn't for the faint hearted :-) I don't believe in quick and dirty approaches to handling intrinsically complicated issues. At the same time, I am not an ivory tower researcher either! When I hear many presentations at various conferences and meetings like the Hadoop User Group (HUG), I have a tough time making sense of what is going on given the high level nature of what is being presented with no serious attempts being made to compare what is proposed with what has been done before and about which much more is known.

Of course, NoSQL systems aren't the only context in which such things have happened in the past. A great number of people have talked about optimistic concurrency control and recovery without much of the details really having been worked out [3]. Even now some of the so-called NewSQL systems' designers make some tall claims about how traditional recovery isn't needed and that they can get away without logging while still supporting SQL, etc. One has to quiz them quite a bit to discover that they do in fact do some bookkeeping that they choose not to describe as logging and/or that they don't support statement-level atomicity even though they claim to support SQL and SQL requires it!

For some people, it might be very tempting to think that the NoSQL applications are so much different from traditional database applications that simple things are sufficient ("good enough" being the often used phrase to describe such things) and that overnight mastery of the relevant material is possible. Even in the Web 2.0 space, if the application programmers are not to go crazy, more of the burden has to be taken up by the designers of the NoSQL systems. A case in point is how the Facebook messaging system designers decided that the eventual consistency semantics is too painful to deal with and hence they chose to go with HBase [15]. To begin with, if the NoSQL systems have vague semantics of what they support and subsequently, as they evolve, if such things keep changing, users will be in big trouble! Also, with no standards in place for these systems, if users want to change systems for any number of reasons, applications might require significant rewriting to keep end user semantics, whatever it is, consistent over time.

8. SUMMARY

I fully realize that there are a variety of NoSQL systems and that there are many differences between them with respect to the functionality that they provide and the technologies that were invented/leveraged/implemented to realize that functionality [18]. While not all the points that I have made in this paper would necessarily apply to every one of those systems, my feeling is that every point I have made would apply to at least a reasonable subset of the systems. I hope the designers and users of NoSQL systems would try to benefit from the contents of this paper.

Hopefully, more of the work of the past decades would be fully leveraged to build more industrial-strength and easier to use NoSQL systems for everybody's benefit!

In the past, when alternatives to relational technology got some momentum, native implementations of such alternatives were developed and such systems got some popularity, RDBMS vendors and researchers started extending RDBMSs with some of the features of the alternatives. A couple of examples are the object-oriented and XML extensions that were made to RDBMSs. We are likely to see similar developments with some of the NoSQL features being incorporated in RDBMSs.

In the past, we have seen certain highly hyped areas of database management, which caused many new products or extensions to existing products to be developed, subsequently not living up to the hype in terms of actual adoption of such technologies by users or vendors making enough money from the investments that were made. This was the case with OODBMSs and XQuery.

At this point, there seems to be lot more momentum behind the NoSQL phenomenon compared to the past when at times a few alternatives to RDBMSs attained some popularity. At least with respect to the startup scene in Silicon Valley and from the perspective of the Web 2.0 style companies, the former appears to be the case. Only time will tell whether this is a passing fad or a long term phenomenon!

9. ACKNOWLEDGMENTS

I wish to express my sincere thanks to my past and present IBM research and product collaborators during three decades who have helped me gain insights into various systems' internals and the rationale behind the design choices that were made.

10. REFERENCES

[1] Choy, D., Mohan, C. *Locking Protocols for Two-Tier Indexing of Partitioned Data*, Proc. International Workshop

on Advanced Transaction Models and Architectures, Goa, August-September 1996.

[2] Mohan, C. *Interactions Between Query Optimization and Concurrency Control*, Proc. 2nd International Workshop on Research Issues on Data Engineering: Transaction and Query Processing, Tempe, February 1992.

[3] Mohan, C. *Less Optimism About Optimistic Concurrency Control*, Proc. 2nd International Workshop on Research Issues on Data Engineering: Transaction and Query Processing, Tempe, February 1992.

[4] Mohan, C. *Repeating History Beyond ARIES*, Proc. 25th International Conference on Very Large Data Bases, Edinburgh, September 1999.

[5] Mohan, C. *The NoSQL Hoopla … What is NonsenSQL about it,* Series of blog posts, http://cmohan.tumblr.com/, April 2012.

[6] Mohan, C., Barber, R., Watts, S., Somani, A., Zaharioudakis, M. *Evolution of Groupware for Business Applications: A Database Perspective on Lotus Domino/Notes*, Proc. 26th International Conference on Very Large Databases, Cairo, September 2000.

[7] Mohan, C., Haderle, D. *Algorithms for Flexible Space Management in Transaction Systems Supporting Fine-Granularity Locking*, Proc. 4th International Conference on Extending Database Technology, Cambridge, March 1994.

[8] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P. *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging*, ACM Transactions on Database Systems, Vol. 17, No. 1, March 1992.

[9] Tweed, R., James, G. *A Universal NoSQL Engine, Using a Tried and Tested Technology*, http://www.mgateway.com/docs/universalNoSQL.pdf, 2010.

[10] Wikipedia. *NoSQL,* http://en.wikipedia.org/wiki/NoSQL

[11] Lamb, C. *Oracle NoSQL Database*, Presentation at International Workshop on High Performance Transaction Systems (HPTS), Asilomar, October 2011, http://hpts.ws/sessions/cwl-hpts-for-website.pdf

[12] Marcus, A. *The NoSQL Ecosystem*, In The Architecture of Open Source Applications, A. Brown and G. Wilson (Eds.), 2011, http://www.aosabook.org/en/nosql.html

[13] Marcus, A. *The NoSQL Ecosystem*, Presentation at International Workshop on High Performance Transaction Systems (HPTS), Asilomar, October 2011, http://hpts.ws/sessions/nosql-ecosystem.pdf

[14] Ellis, J. *Apache Cassandra Present and Future*, Presentation at International Workshop on High Performance Transaction Systems (HPTS), Asilomar, October 2011, http://hpts.ws/sessions/Cassandra_HPTS_2011.pdf

[15] Muthukkaruppan, K. *Storage Infrastructure Behind Facebook Messages*, Presentation at International Workshop on High Performance Transaction Systems (HPTS), Asilomar, October 2011, http://mvdirona.com/jrh/TalksAndPapers/KannanMuthukkaruppan_StorageInfraBehindMessages.pdf

[16] Cockcroft, A. *Netflix Global*, Presentation at International Workshop on High Performance Transaction Systems (HPTS), Asilomar, October 2011, http://hpts.ws/sessions/GlobalNetflixHPTS.pdf

[17] Anand, S. *Netflix's Transition to High-Availability Storage Systems*, Netflix Document, October 2010, https://bitly.com/bhOTLu

[18] Edlich, S. *List of NoSQL Databases*, http://nosql-database.org/

[19] Srinivasan, V., Bulkowski, B. *Citrusleaf: A Real-Time NoSQL DB which Preserves ACID*, Proc. 37th International Conference on Very Large Databases, Seattle, August 2011.

[20] Wayner, P. *7 Hard Truths about the NoSQL Revolution,* InfoWorld, 16 July 2012.

[21] Popescu, A. *myNoSQL – NoSQL Databases and Polyglot Persistence: A Curated Guide*, http://nosql.mypopescu.com

[22] Edlich, S. *The State of NoSQL*, November 2012, http://www.infoq.com/articles/State-of-NoSQL