SOCIAL BOOKMARKING OVER NOSQL. WE NEED SPEED.

Ljupco Jovanoski
Faculty of Computer Science and Engineering
Skopje, Macedonia

Vladimir Apostolski
Faculty of Computer Science and Engineering
Skopje, Macedonia

Dimitar Trajanov
Faculty of Computer Science and Engineering
Skopje, Macedonia

ABSTRACT

Becoming the backbone of some of the web's biggest social networks, and few others systems holding a large amount of data, the NoSql concept stands its ground as the only alternative to storing data beside the SQL approach. Big web companies like Google, Amazon and Facebook developed non-relational databases that sacrifice consistency for availability, scalability and performance. We have developed a social bookmarking service based on NoSql concept, looking for performance, since expecting a large amount of data in our hands. A system providing its users with an easy way to organize, tag and share web content of interest. Additionally, generating recommendations and groups of similar web content, based on interests shown while using the bookmarking service. In this paper we concentrate on building a solid ground for performance comparison of two web applications, providing the same social bookmarking service, using a different concept to store its data. The expectations are that the NoSql based service will provide us with better performance that the SQL supported service. We concentrate on the potential performance benefit, as well as the possibility to gain additional advantage by being able to scale our data storage on multiple servers.

I. INTRODUCTION

When it comes to social networks, its users always cared about speed, and it was never any different since the trend started. The choice of transferring the social aspect of life on one of the many social network has been made, so it seems that this huge responsibility has been delegated to the service provider, to take care of its user's needs, mainly recognized as performance and accurate data, as well as availability at any given time. Not only social networks will be covered over the content of this paper, since the main goal is to observe web systems that handle huge amount of data. For a long time, nobody was ambitious enough to ask what needed to be asked long time ago – is there an alternative to the SQL approach? Over time, SQL walked its obstacle-free way, gaining more and more thrust with every SQL-based relational database management system being developed. Today, SQL stands tall, still being the first and obvious choice when it comes to system development in need of data storage. But today brings a new challenge, removing the rug underneath SQL's feet, forcing it's "soon to be ex" clients to search for the long forgotten need of an alternative in order to handle the humongous amount of data:

- one billion searches on Google per day [1]
- 60 million statuses updated every day (not counting photos upload, likes and other types of interaction available on Facebook) [2]
- 175 million tweets per day [3]

So, this being the statistics nowadays, handling this amount of data is impossible with SQL, if the provider's main concern is performance. Gaining a high degree of horizontal scalability is another issue that SQL does not respond very well to [4].
The question came up in 2004 and who else but Google to be the first one to ask and answer it. "BigTable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers" – so they said when they decided to come out to the world in 2006, full-feature paper, describing the surface of BigTable, keeping most of the fun part to themselves, as well as the code [5]. Amazon basically did the same thing with their Amazon DynamoDB implementation of the NoSql concept, with the difference of making profit of it by offering it as a service. "DynamoDB will automatically spread the data and traffic over a suitable number of servers using solid state drives, allowing predictable performance"[6] [7]. Facebook was lunched at 2004, but since they did not think big at first, it was not a concern for them at the time. Since September 26, 2006 and the opening of the Facebook service to everyone above the age of thirteen, the drastic growth of their social network, forced them to think about alternative approaches to their MySQL foundation [8]. They came up with Cassandra, which represents a structured key-value store with tunable consistency, which is a big step up from "eventual consistency", since now a choice is offered between data being "eventually consistent" or "strongly consistent". So, Facebook having a role model and someone to look up to, developed Cassandra and used it for their Inbox Search feature. Not long after that, they released it as an open source project on Google Code. Since February 2010 Apache adopted Cassandra as a top level project and today Cassandra enjoys a healthy community around it [9].
A trend has developed, so the concept was out there for everyone else to embrace it, and many social networks did this, taking advantage of the ability to scale their data storage over multiple nodes, thus gaining performance over a large data set. However, everyone that took the step towards the alternative, had to sacrifice few things from both developer's and client's perspective. The development aspect suffered a loss of rich query language, the relations between entities, thus a new DB structure is needed to represent entities, compromising data consistency and causing data redundancy. Despite these disadvantages of the newly born NoSql concept, it seems that it found its place when it comes to data storage. The NoSql movement showed its strength back in 2009 and since then is growing rapidly in terms of implementations and community. Looking few years back, we can see that the most respectable social networks out there

decided to take this big step towards the alternative to SQL [10] [11]. Thus, after developing our social bookmarking service powered by SQL Server 2008, we dedicated some time to research, to determine if there is a better way to store big amount of data.
Our intention is to roughly estimate the potential of NoSql, powering our social bookmarking service and every other service provided by the system we developed. The initial assumption is that we will gain performance, as well as the ability to scale horizontally.

II. THE IDEA –BIRTH, MEANING, DEVELOPMENT

The definition of our system covered an easy to use bookmarking service and a few other features that interfere with the concept of social networks, like sharing content, forming open groups of interest, generating suggestions based on user interests and following users with similar interests. We tended to achieve high degree of personalization by determining user's interests by the actions he takes while using our service. Conducting our research about the competition on the web, we came upon few web sites that introduced us with features we wanted to see in our system, being under the impression our users would benefit from them. However, despite the things we found "missing", we tended to look for something the competition was missing, led by the idea "let the information find you, not the other way around". We then decided that the information in our system needs to hold more meaning and value than plain text, in order to find some kind of connection with the interest shown by the users of our bookmarking service. Thus, we turned towards a semantic approach, which we incorporated with the help of an external knowledge source exposed as a web service – the OpenCalais Web Service. We use the OpenCalais Web service to retrieve semantic keywords over a given web content being marked [12]. We consider this to be the right way to offer more precise web content to our users. From this point on, two research directions developed: how to present the user with the information he needs (semantic data annotation), and how to do the same faster (consider SQL alternative). This paper will cover the later, presenting the process of SQL to NoSql migration and the research which provides the final answer regarding the obtained benefits, and the lost benefits as well. Attempts of comparison between the two concepts in terms of performance were made, since the NoSql movement started, showing great performance improvement [13] [14].

III. DATABASE SWITCH

Adopting the alternative of the relational data storage systems, in order to gain advantage in terms of scalability, availability and performance, we lost something that comes in handy during development [15]. Relations between entities, the rich query language including join, order by, group by statements, as well as the ability to define a firm policy over the entire data storage in order to gain data consistency and remove any possibility of data redundancy [16]. What the migration forced us to do, to overcome the NoSql approach disadvantages was the following:

- Redesign DB structure
- Hold the entire entity as one record, instead of breaking it in multiple tables like in SQL
- Overcome lack of rich query language by generating additional collections holding statistics
- Multiple collections hold same information
- Partially resolve data consistency issues at application level
- Make peace with being "eventually consistent"

Observing the bookmark segment of the SQL implementation, we can see that a single bookmark record is spread across five different tables holding the basic information about the web content being marked, the semantically annotated tags relevant to the same, along with their relevance bound to the web content and the user of the bookmarking service (Fig. 1). Having the ability to define relations between tables, and the powerful query language in perspective, these five tables are enough to cover the stream of latest bookmarks made by using our bookmarking service, stream of bookmarks made by the user himself, as well as retrieving global trends and generating recommendations dependent on single user interests.

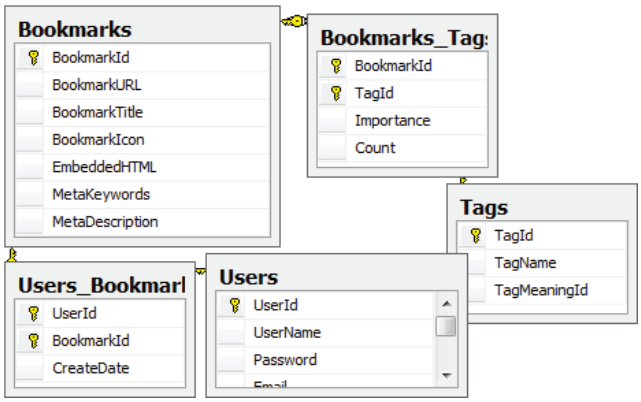

Figure 1: Database diagram showing tables relevant to the bookmarking process, relations between them and data they hold.

Multiple attempts to translate the SQL DB model to NoSql DB model were made. However, because of MongoDB's flexibility in terms of change of entity definition, we spent zero time on data migration during minimal document corrections [17]. We switched the SQL tables with collections, every SQL row is represented as one BSON document, containing key value pairs, representing strongly typed domain model entities. The final approach we came up with was to have all basic bookmark information in one entity – document, along with the tags related to the web content and short information regarding the users that have marked

the same web content (Fig. 2). The tag collection within every bookmark document holds minimal but sufficient information, covering the tag name as well as the tag meaning and the relevance in regard to the web content being marked. This collection covers the stream of latest bookmarks and the ability to produce the statistics for the most popular posts on global level. However, it does not have the potential to cover stream of user bookmarks, and to determine the user interests, thus we cannot rely on it to recommend content of interest to each user. To overcome this issue, an additional collection was created (UserBookmarks), in order to provide the system with the ability to produce the stream of single user bookmarks, as well as the possibility to retrieve his interest based on his bookmarking history, generating a list of bookmarks made by other users, which might be of interest to him.

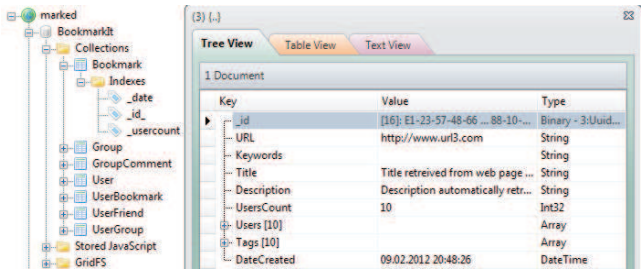

Figure 2: Database diagram showing tables relevant to the bookmarking process, relations between them and data they hold.

The downside we face here is the data duplication leading to failure in the redundancy aspect, as well as disruption of data consistency. However, this risks and disadvantages did not come to perspective at this stage of the NoSql implementation, since they were expected and accepted during the initial research for migration to alternative data storage.

When it comes to the base user information, not much has changed in the model. However, the analysis showed that every User document can hold information about the user's actions, in terms of bookmark activity, as well as activities connected to group creation and participation. Thus, an additional two properties were added to the User model, holding the number of individual user posts and number of groups in which the user participates, enabling a fast way to generate global trends within our application. The groups of interests are organized on database level identically as the bookmarks, having an additional collection holding the comments of the group's users. The social aspect of the service, represented as following users sharing same interests, resulted in an additional UserFriends collection in the database model, holding information about user's relations with other users of the social bookmarking service. A single Message collection is also added to the database model to represent users interaction implemented as a simple message sharing system. Despite being present in both bookmark and group models, the statistics for keyword representation and trend are available through the Tags collection, created especially to have an easy and precise way to cover the application's trend, and determine the target group. Additionally, it provides a fast search through the closed set of tags covered by the application, enabling tag suggestions to the users, while marking and organizing web content.

IV. System Architecture

The system is realized as a web application with a non-relational database system at the backend. What used to be a Microsoft SQL Server 2008 instance was replaced with a NoSql implementation MongoDb. MongoDB, a cross-platform NoSQL database, is the fastest-growing new database in the world. MongoDB provides a rich document-oriented structure with dynamic queries that you'll recognize from RDBMS offerings such as MySQL [18]. Before replacing our SQL implementation with NoSql implementation, we conducted a research that ended with MongoDB imposing as the primary candidate for data storage. MongoDB has developed a healthy community in the past couple of years. Additionally, the agility that it provides during the development process, built-in scalability, indexing, JSON-like documents and cross-platform nature, made the decision easy [19]. The multiple MongoDB drivers for .NET just went along with the choice. We used "Samus MongoDB - CSharp" to connect to the document – oriented database. The frontend is implemented as ASP.NET MVC 3 web application (Fig. 3), updated from the previous web solution being built as ASP.NET 3.5 Web application. This is another step forward, towards new technologies and enhanced patterns for software development, since being up to date is just another way to present system quality.

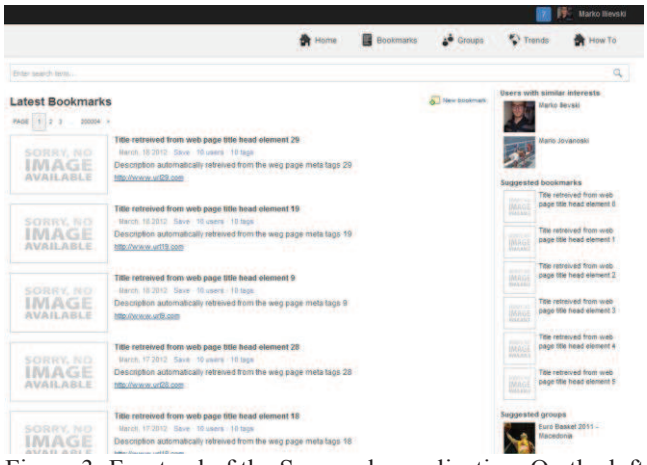

Figure 3: Frontend of the Semmarks application. On the left is the list of latest bookmarks.The right side the recommendations regarding people, bookmarks and groups. The top holds the navigation menu, as well as the search interface and the user's information.

The frontend provides the user with a list of the latest bookmarks, web content marked by other people, as well as his own posts. Alongside, are the recommendations of people sharing same interests, and groups and bookmarks related to those interest.

The easy to use bookmarking process is possible through the browser bookmarklet (Fig. 4), a small piece of JavaScript embedded into any browser, which allows the user to bookmark the current page loaded into the browser.

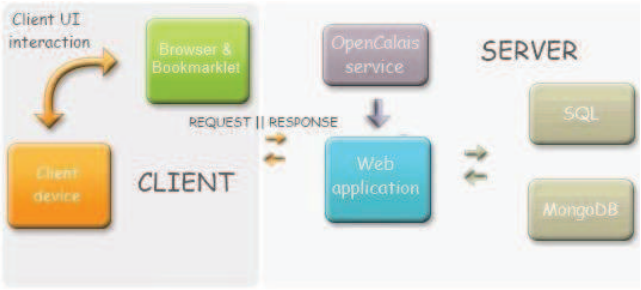

Figure 4: Application Architecture: components of the engine for storing bookmarks. Client interaction remains the same, the change reflected on server side, by changing the data storage provider from SQL to MongoDB.

The advantage the bookmarklet holds over browser extensions lies in its cross-browser compatibility and consequently, there is no need to develop multiple versions for each browser.

After the user initiates the bookmarking process, the URL that the browser's address bar holds is sent to OpenCalais by an asynchronous call, resulting in JSON response, containing the recognized keywords within the web content, along with their meaning and relevance score. The tags are displayed to the user, at which point he can provide tags on his own to complete the bookmarking by saving, causing a single bookmark entity to be sent back to the server, and be written in the database as a single BSON document in the Bookmarks collection. This document holding single web content reference contains its base information like URL, title and metadata contained in the resource's HTML head section. Probably the most important data contained in this JSON-like document, is the list of tags, representing entities found within the web content, along with their relevance to the content in question, as well as their meaning, information describing the keyword, its origin, its meaning in the context, bringing even more knowledge to the system, enabling for a precise content suggestion. The most valuable keywords are the ones retrieved from the OpenCalais web service, semantically annotated entities holding information about the number of occurrences within the given article and relative importance. We combine these two segments to determine if the keyword is relevant enough to be included in our data storage. This way we prevent database flooding with irrelevant tags. Providing information about the information is a huge step forward from the traditional keyword-analyzing techniques, empowering us with a more precise way to determine user interests, thus achieving high level of personalization by delivering the right information to the user. An additional update to a specific User document is made corresponding the logged in user information, incrementing the value of the properties that hold the numbers of the user's activities. Additionally, a new record is added to the UserBookmark collection, making it possible for the post to appear in the stream of user bookmarks. Having in consideration that the keywords attached to the web content are of crucial meaning to defining a closed set of user interests, a Tags collection for the same is being updated with every bookmarking activity, and every group creation. The Tags collection contains every keyword, regardless of its source: manual user input or OpenCalais service response. This is a consequence of the non-relational nature of the entire NoSql concept, regardless of the implementation, forcing us to duplicate data among multiple collections in the database, as well as implementing minor statistics helpers in order to overcome the loss of group by, order by and join statements.

V. Testing Process

Our team set up the environment to perform the test, not taking sides in the SQL versus NoSql competition. We managed to disabled the caching on the SQL instance, to rely on SQL itself for performance. Because of our expectance to get the most activity from our bookmarking service, we decided to profile over the page displaying the stream of latest bookmarks. We also decided to relieve both SQL and MongoDB from additional processes using it, in order to see their response over a single query at any given moment. Thus, we removed every single additional request on the page we are profiling (logged in user info, bookmark, group and user suggestions), leaving only the stream of latest bookmarks to be profiled. Finally, we had the same conditions applied to both systems, expecting that we will come in possession of data showing the proportion between SQL Server 2008 and MongoDB performance. Hopefully, the results collected from the tests in development environment apply in production environment as well. A stored procedure is used by SQL to retrieve the stream of latest bookmarks, implementing standard paging, sorting the bookmarks by date of creation and a return parameter holding the number of records satisfying the given criteria. MongoDB implements the same logic, using the C# MongoDB driver at data layer. We are measuring the precise time from the moment the connection to the database is established, the data retrieval, to the moment the connection to the database is closed. The profiling is being conducted by initiating 30 different requests to each system, by changing the page parameter sent to the server as a part of the request. Pages requested over which the profiling will be executed are shown in the following intervals: [1-10], [50-59], [90-99]. Regarding the content that is being inserted in the databases, it's different in both cases due to the significant change in the data model while migrating from SQL to MongoDB. However, the same amount of content is provided to both SQL Server 2008 and MongoDB and it covers the basic bookmark parameters like web page title, web page URL, the description and the keywords retrieved from the meta tags of the web page, list of 10 keywords related to the web content and a list of 10 users which already bookmarked the web page. In MongoDB, all this data is represented as a single document in the Bookmark collection. In SQL Server 2008, the data being inserted is separated in the tables shown in Fig. 1, where the Bookmarks

table holds the title, URL, description and keywords of the web page, while the users and tags retrieved from the web content are located in Users_Bookmarks table and Bookmarks_Tags.

A. *Testing Environment*

Using a simple profiling tool available at Google Code [20], installed over both web applications through the NuGet, a Visual studio extension, making it easy to install third-party libraries [21]. The hardware configuration consists of Dell Latitude E6520, with i5-2410M CPU @ 2.3 GHz, 8 GB RAM, Windows 7 Ultimate operating system. Both web applications hosted on local machine on IIS 7.5, under the same application pool with .NET Framework v.4.0.30319. The focus is on SQL Server 2008 instance and the MongoDB instance, running as a Windows Service on port 27017 [22].

VI. RESULTS

Four scenarios were presented by our team, expecting that the given conditions will cover every aspect of the research. The attempt is to profile the stream of latest bookmarks on both SQL and MongoDB based web applications. For this purpose, a console application was developed to make the dummy data insert, using it at four different stages to provide half million, a million, three million and five million records in both SQL Server 2008 and MongoDB instances. The initial idea for the scenarios was to find common grounds at the first stage and work our way up to a point where the proportion between performances will prove out initial thesis right or wrong.

A. *Results at Half Million Records*

It seems that at this point of the research, we found a common ground for both instances, showing similar execution times. SQL Server 2008 presented an average of 32.5 milliseconds per page request and MongoDB presented slightly lower results with the average of 19.76 milliseconds per page request (Fig. 5). Despite the fact that the NoSql implementation presents a little advantage in performance, the loss of convenient relations between entities and the ease of querying, is not supported by this minor benefit in performance.

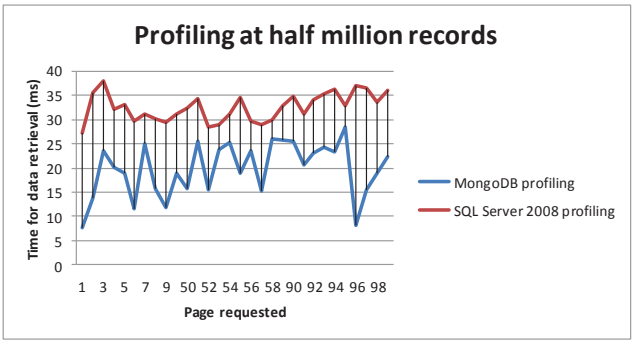

Figure 5: The performance test results at half million records in both SQL Server 2008 and MongoDB. SQL handles itself quite good at this stage of the research, as expected. However, MongoDB instance shows slightly better performance.

B. *Results at One Million Records*

At this point of the research, it seems like a tendency has developed, going along with our initial thesis. The results at one million records show 27.723 milliseconds needed for a request to be served by the MongoDB instance, against 82.16 milliseconds for a request served by SQL Server 2008 (Fig. 6). Compared to the previous profiling session at half million records, SQL made a significant jump in performance, while MongoDB went slightly up on the time scale.

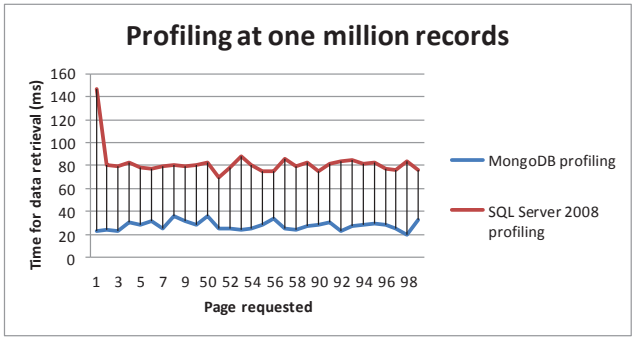

Figure 6: Chart showing the test results at one million records. SQL raises its response time.

C. *Results at Three Million Records*

The trend continues. After inserting three million records in both SQL Server 2008 and MongoDB, the profiling resulted in an average of 25.77 milliseconds per page request on MongoDB's side and a high raise to 370.53 milliseconds per page request on SQL's side (Fig. 7). A definite advantage in performance is noticeable right about now, and a final blow to the SQL instance will be given at five million bookmark records.

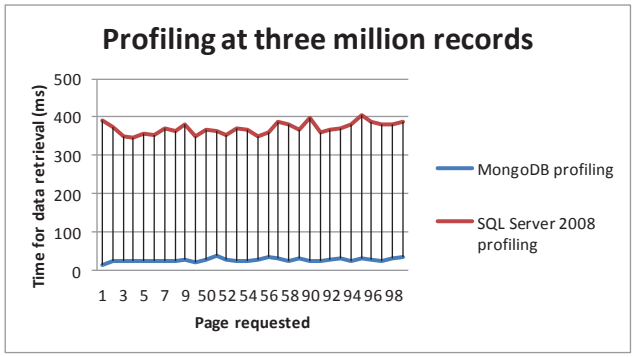

Figure 7: Chart showing the test results at three million records. SQL drastically rising, while MongoDB remains at impressive 25.7 milliseconds.

D. *Results at Five Million Records*

At this point of the research, we got the results that will fully support our initial thesis, the gain of performance by switching to the alternative of the long trusted relational database management systems – the "Not Only SQL" concept. A massive difference it performance supported by the comparison of the profiling results at five million records:

33.296 milliseconds per request for MongoDB against 570.01 milliseconds per request for SQL Server 2008.

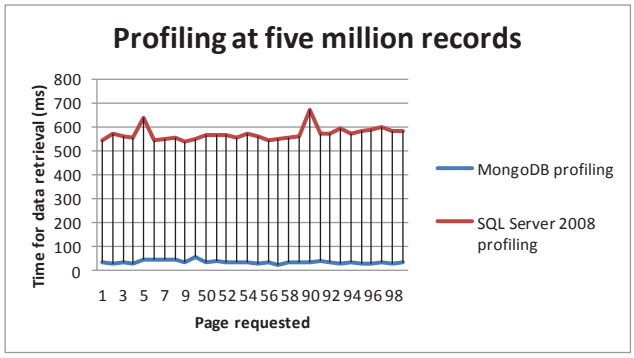

Figure 8: Chart showing the test results at five million records. MongoDB shows the necessity of an alternative to the relational database approach.

VII. Conclusion

Based on the presented results, our team came to the conclusion that advantage can definitely be taken from MongoDB or any other NoSql implementation for that matter. When it comes to a social bookmarking service, which extends its appetites towards social network features, as well as semantic annotation of web content, performance for the users and an easy way to scale the data storage from developer's point of view, is the way to go. Main factor that contributed to this decision was the noticeable difference in performance over five million records. Additionally, the real power of NoSql is his ability to scale out easily across multiple servers, a feature that comes as a contradiction to the relation aspect of the relational database management systems. However, both development and customer sacrifices are inevitable due to absence of relations between entities and depletion of the rich query language that SQL offers. SQL still remains as the obvious and logical choice in systems that required data storage and handle relatively small amount of data. However, with the trend that has developed in recent years, the migration of the social aspect of life on the web, it seems that SQL no longer has the entire market for itself. NoSql becomes even more appealing with the fact that it remains open-source and the tendency to develop a healthy community over every implementation of the concept. Having the policy of the social networks in consideration, eventual data consistency, non-relational nature and abandoning the set of extremely useful SQL statements, seems like a fair trade for higher degree of availability, performance and scalability.
In future, a possibility remains to improve the database structure, and scale the data over multiple nodes, taking full advantage of NoSql. Our team latest interests are related to a different NoSql database types, analyzing their benefits over MongoDB, so a different NoSql implementation is also a possibility, opening a new window of opportunities for research and improvement.

References

[1] *Google Searches per Day*. August 2011

[2] Anson Alexander. "*Facebook User Statistics 2012 [Infographic]*". February 2012

[3] Shea Bennett. "*Just How Big Is Twitter In 2012? [Infographic]*". February 2012

[4] Rick Cattell. "*Scalable SQL and NoSQL data stores*". December 2010.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber. "*Bigtable: A Distributed Storage System for Structured Data*". OSDI 2006

[6] Vogels, Werner (2012-01-18). "*Amazon DynamoDB – a Fast and Scalable NoSQL Database Service Designed for Internet Scale Applications*". January 2012.

[7] Carolyn Abram. "*Welcome to Facebook, everyone*". September 2006.

[8] Clark, Jack. "*Amazon switches on DynamoDB cloud database service*". ZDNet. 2012.

[9] *The Apache Cassandra Project* . 2009.

[10] "*Looking to the future with Cassandra | Digg About*". September 2009.

[11] "*Cassandra @ Twitter: An Interview with Ryan King*". February 2010

[12] *Open Calais: How Does Calais Work?* . 2008.

[13] Michael Kennedy. "*The NoSQL Movement, LINQ, and MongoDB - Oh My!*". April 2009

[14] Michael Kennedy. "*MongoDB vs. SQL Server 2008 Performance Showdown*". April 2009

[15] Kai Orend. "*Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer*". April 2010

[16] Rick Cattell. "*Will NoSQL Databases Live Up to Their Promise?*". February 2010.

[17] Kristina Chodorow, Mike Dirolf. "*MongoDB: The Definitive Guide*". September 2010

[18] Eelco Plugge, Tim Hawkins, Peter Membrey. "*The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*". 2010

[19] Christof Strauc. "*NoSQL Databases*". February 2011

[20] *A simple but effective mini-profiler for ASP.NET and WCF - Google Project Hosting* . *2010.*

[21] *Nuget: Visual Studio extension that makes it easy to install third-party libraries. 2010.*

[22] *Windows Service – MongoDB. 2009.*