Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departmento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação
Mestrado Acadêmico em Sistemas e Computação

TODO::CHANGE::Guidelines for database transitioning on production environments

Fabio de Sousa Leal

Natal-RN
Julho de 2015

Fabio de Sousa Leal

TODO::CHANGE::Guidelines for database transitioning on production environments

Proposta de dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Mestre em Sistemas e Computação.

Linha de pesquisa:
Engenharia de Software

Orientador

Martin A. Musicante

PPgSC – Programa de Pós-Graduação em Sistemas e Computação
DIMAp – Departamento de Informática e Matemática Aplicada
CCET – Centro de Ciências Exatas e da Terra
UFRN – Universidade Federal do Rio Grande do Norte

Natal-RN

Julho - 2015

Dissertação de Mestrado sob o título *TODO::CHANGE::Guidelines for database transitioning on production environments* apresentada por Nome completo do autor e aceita pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Martin A. Musicante
Presidente
DIMAp – Departamento de Informática e Matemática Aplicada
UFRN – Universidade Federal do Rio Grande do Norte

Marcia Jacyntha Nunes Rodrigues Lucena - Professora Adjunta
Examinador
DIMAp – Departamento de Informática e Matemática Aplicada
UFRN – Universidade Federal do Rio Grande do Norte

Thaís Vasconcelos Batista - Professora Adjunta
Examinador
DIMAp – Departamento de Informática e Matemática Aplicada
UFRN – Universidade Federal do Rio Grande do Norte

Natal-RN, data da defesa (dia, mês e ano).

Homenagem que o autor presta a uma ou mais pessoas.

Agradecimentos

Agradecimentos dirigidos àqueles que contribuíram de maneira relevante à elaboração do trabalho, sejam eles pessoas ou mesmo organizações.

Citação

Autor

TODO::CHANGE::Guidelines for database transitioning on production environments

Author: Fabio de Sousa Leal
Supervisor: Martin A. Musicante

Abstract

Component-based Software Engineering (CBSE) and Service-Oriented Architecture (SOA) became popular ways to develop software over the last years. During the life-cycle of a software system, several components and services can be developed, evolved and replaced. In production environments, the replacement of core components, such as databases, is often a risky and delicate operation, where several factors and stakeholders take place.

Service Level Agreements (SLA), according to ITILv3's official glossary, is "an agreement between an IT service provider and a customer. The agreement consists on a set of measurable constraints that a service provider must guarantee to its customers.". In practical terms, it is a document that a service provider delivers to its consumers with minimum quality of service (QoS) metrics.

This work assesses and improves the use of SLAs to guide the transitioning process of databases on production environments. In particular, in this work we propose SLA-Based Guidelines/Process to support migrations from a relational database management system (RDBMS) to NoSQL one. Our study is validated by case studies.

Keywords: SLA, Database, Migration.

List of Figures

List of Tables

Lista de abreviaturas e siglas

Lista de símbolos

Lista de algoritmos

Sumário

1 Introduction

Cloud computing became a reality over the last years, and many companies are now moving their data-centers to the cloud. A concept that is often linked with cloud computing is Infrastructure as a Service (IaaS): the computational infrastructure of a company can now be seen as a monthly cost instead of a number of different factors. Recently, a number of organizations started to replace their relational databases with hybrid solutions (NoSQL DBs, Search Engines, ORDBs).

These changes are motivated by (*i*) performance improvements on the overall performance of the applications and (*ii*) inability to a RDBMS to provide the same performance of a hybrid solution given a fixed monthly infrastructure cost.

However, not always the companies can exactly measure beforehand the future impact on the performance of their services by making this sort of technological changes (replace RDBMS by another solution).

In a production environment, it is necessary to assure that a database transition will actually bring benefits to the overall performance of the application. To avoid external threats and unknown risks, a database transition must be made in a pragmatic manner on each step of the transition: from the initial hypothesis to the deployment of the new architecture.

1.1 Problem

The decision to migrate (part of) an application from RDBMSs to NoSQL[1] alternatives , such as Search Engines or Graph Databases, is justified when the alternatives have better performance/manutenability than the classic RDBMSs and the cost to have

[1]NoSQL stands for **Not Only** SQL (QI et al., 2014). Some authors also refer to NoSQL as "Non-SQL", "NoREL" (non relational) or even "New SQL". (NOSQL-ORG, 2015) defines it as being *Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable.*

a similar performance on the current database architecture is significantly higher.

Several steps are needed in the process of replacing a relational database by a NoSQL alternative. An initial step on transitioning processes could be to map the relational schema into the new kind of document. Some works, such as (LOMBARDO; NITTO; ARDAGNA, 2012) (ZHU; WANG, 2012) address this sort of problem.

Schema mappings across different technologies are, however, very particular to each application. (BAHL, 2014) shows, for example, how a MySQL schema may be represented on MongoDB and Neo4j, both NoSQL DBs.

The need for a transition may be characterised by showing that the current database infrastructure is breaking some of the requirements of an application, such as the speed of an operation. This may be done by the result of a DB Benchmark or the execution of an automated test, for example.

Several Benckmarking Frameworks, such as TPC-H (COUNCIL, 2008), can be used on this step. Automated tests can make use of testing libraries/frameworks, as jUnit (MASSOL; HUSTED, 2003) and RSpec (CHELIMSKY et al., 2010) or can be implemented from scratch using popular programming languages, such as Java and Python.

Database transitions generally follows non-standardized methods and processes may differ significantly in each application, as revealed on (LEAL; MUSICANTE, 2015).

In this work, we assess how Service-Level Agreements (SLAs) can be used to help the migrations from RDBMSs to NoSQL solutions and propose a set of Guidelines based on SLAs to justify and assess the transitions from RDBMs to NoSQL alternatives.

1.2 Proposed solution

Changing the database layer of an application is a decision that requires ample thinking, extensive analysis of the risks & impacts and a large number of experiments and benchmarks.

If a database transition doesn't follow a pragmatic process, bugs may be introduced on the application, rework may be necessary and even data the existing data may be corrupted.

In this work we propose a set of guidelines based on Service Level Agreements (SLAs) to assist the transition of the database layer on applications. An example is given to clarify

how this process can be used:

1.2.1 Example: Database transitioning on a simple TO-DO list application

Suppose that a simple to-do list application was built on the top of MySQL (RDBMS) with the following features:

- Create, Update, Retrieve and Delete (CRUD) of tasks;
- Search tasks by query string.

An SLA could then be established for this application. One of the objectives of this SLA could be its the search speed. i.e: *Every search operation must be executed in less than 2 seconds.*

As the number of tasks on the database grow, users may claim that the speed of task searching has degraded. In fact, this is an expected scenario, as the number of DB records to be analyzed also grows.

The developers of the application could then claim that a database transition might be needed, as a *Search Engine* is more sophisticated than MySQL to perform full-text search features.

From the SLA of the application, we could then build a transitioning process that performs benchmarks with the current technology (MySQL) and the proposed technology (a Search Engine) to verify if a database transition is really needed.

The steps & guidelines proposed by this work are detailed and extensively discussed on Chapter 3

1.3 Contribution

In this work, the main contributions are:

- Proposition of a set of Guidelines based on SLAs to justify and and guide the transitions from RDBMs to NoSQL DBs;
- A Batch PDF-Tokens matcher, available on (PDFTOKENSMATCHER, 2015);

- A Systematic mapping study on "Using SLA to guide database transition to NoSQL on the cloud", available on (LEAL; MUSICANTE, 2015);
- Proof of concept for our guidelines in the form of Migration scenarios.

1.4 Structure of this work

On Chapter 2 we present a detailed view on each concept that surround this work and explain how each of these concepts are linked with our work.

On Chapter 3 we extensively detail our problem and the proposed solution.

On Chapter 4 we present how the guidelines defined on Chapter 3 might be used on a RDBMS to NoSQL transition.

On Chapter 5 we present the next steps that are possible from this work. The references that were used on this work are available after Chapter 5.

2 Bibliographic Review

The goal of this chapter is to explore the main concepts that are related to this work. We begin the section presenting popular concepts, such as Cloud Computing, *Everything as a Service* strategy, NoSQL and Service Level Agreements. We also explain how each of these concepts are linked with this work.

2.1 Cloud Computing

Technology evolved with big steps over the last decades. Internet is now a pervasive concept and individuals can be connected virtually everywhere on Earth. (ARMBRUST et al., 2009)

Web applications and IT-based processes followed this evolution and today a number of company rely on software on its production chain. Information Technology can be a competitive advantage of a company, but it **might not** be part of its core business (POWELL; DENT-MICALLEF, 1997). In fact, this is a common scenario on a number of successful companies of the current century, as we might notice.

To avoid losing track of its core business, a number of companies now prefer to outsource (part of) their IT department to other companies (QUINN; DOORLEY; PAQUETTE, 2013). In other words, today it is possible to outsource IT infrastructure, product development and even the entire IT department to other companies.

In the late 60's, former Stanford University professor John McCarthy introduced the concept of time-sharing of computing services in a famous speech on time-sharing systems, as referenced by (RAPP, 2011) and (WIKIPEDIA, 2015). In fact, Prof. McCarthy believed that computer resources would be provided as commodities, like water and electricity.

Several years later, this concept brought to life the notion of *Cloud Computing*, together with new concepts, such as Infrastructure as a Service *(IAAS)*, Platform as a Service *(PAAS)*, Software as a Service *(SAAS)* and Everything as a Service *(XaaS)* (CONNOLY

D. FOX, 2010).

According to (CONNOLY D. FOX, 2010), *Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services.*

(STANOEVSKA-SLABEVA; WOZNIAK, 2009) outlines some of the features of Cloud Computing:

- Cloud Computing is a new computing paradigm.
- The main features of clouds are virtualization and scalability on demand.
- Infrastructure resources (hardware, storage and system software) and applications are provided in X-as-a-Service manner.
- Utility computing and SaaS are delivered in an integrated manner. Computing might be consumed separately.
- Cloud services are consumed either via Web browser or via a defined API.

The "pay as you go" model offered by Cloud providers revolutionized the IT market, enabling companies to consume computing resources that matched its needs. Small companies became more competitive, as there's no more need to build datacentres to scale companies. As professor McCarthy believed, computing resources can now be seen as commodities for a company, just like water and electricity.

2.2 Everything as a Service - XaaS

Service-Oriented Architecture (SOA) defines several concepts of "as-a-service" models. To enumerate a few, it is possible to find mentions to Platform as a Service (PaaS), Infrastructure as a Service (IaaS), Software as a Service (SaaS), Database as a Service (DBaaS), Desktop as a Service (DaaS), Monitoring as a Service (MaaS) and Communication as a Service (CaaS) on the literature.

To summarize all these concepts, a new term arose: Everything as a service (XaaS)(DUAN et al., 2015)(ARMBRUST et al., 2009).

On the context of Cloud Computing, however, three of these concepts are the most relevant, and we define them more precisely:

- Infrastructure as a service (IaaS): It is the most simple kind of "as-a-service" product and is located on the base of the IaaS-PaaS-SaaS Stack (Figure 1). IaaS mostly refers to (Virtual) Machines, Storage Devices, Network Infrastructure and other infrastructural services that are available on Cloud Computing vendors. Some examples of IaaS providers are (EC2, 2015) (RACKSPACE, 2015) and (AZURE, 2015).

- Platform as a Service (PaaS): PaaS refers to the development environments that are available from cloud vendors. PaaSs are composed by a development stack, and generally offer databases, web servers and execution runtime. Examples of PaaSs are (BEANSTALK, 2015), (AZURE, 2015) and (APPENGINE, 2015).

- Software as a Service (SaaS): Software as a Service refers to the applications that run on the cloud: Webmail, CRM, Gaming Platforms, Learning Management Systems, etc. SaaSs, just as IaaSs and PaaSs, generally charge its users a periodic fee. The fee is generally conceived in a pay-as-you-go model, so users get charged in a scalable way.

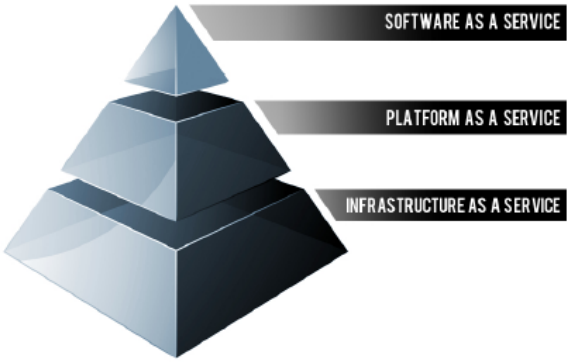

Figura 1: IaaS-PaaS-SaaS Stack (KEPES, 2011).

Since the beginning of the Cloud movement several discussions were held concerning the security of Clouds. (SHINDER, 2010) discusses that the security threats that arise on Cloud Computing are the result of users/enterprise lack of control of the IaaS layer. Not all companies know where their documents/data is physically stored and what are the security instruments that must be used to assure data safety on Cloud environments.

On the base of the pyramid of Figure 1 is located the IaaS - machines, Network Infrastructure and other Simple Services. Above IaaS lies the PaaS layer - It acts as a "middleware" between the SaaS Layer and the IaaS layer, providing the development environments that developers need to deploy applications. On the top of the image is located the SaaS layer.

To have a better understanding of the security concerns on Cloud Computing, a deeper understanding of its architecture is needed. Figure 2 illustrates the reference model for cloud computing.

The IaaS layer is generally composed by five elements: API's, Abstraction, Core Connectivity, Hardware and Facilities. A cloud provider may be vulnerable if a security breach is discovered on its APIs or Abstraction Layer, for example. Another possible vulnerability in cloud providers are outages. In 2014, major cloud providers, such as (EC2, 2015), (AZURE, 2015), (RACKSPACE, 2015) and (APPENGINE, 2015) experienced downtime (PARISEAU; JONES, 2014).

Similar security issues might be found on the PaaS and SaaS layers.

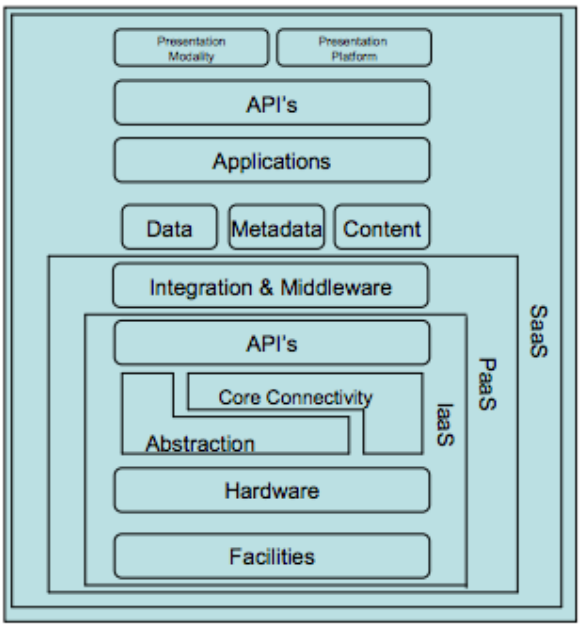

Figura 2: Cloud Reference Model (ALLIANCE, 2009).

2.3 The technological shift

The adoption of cloud solutions is growing fast among organizations (ARMBRUST et al., 2009). Centralized (mostly mainframe) technology is being replaced by distributed and more flexible forms of data storage and processing. This change of paradigm is motivated by the necessity to improve the use of resources, as well as by the increasing velocity in which data is produced.

On the early 90's it was commonplace for every Information Technology (IT) company to have its own Data Center with huge servers and mainframes. IT costs were high,

and high-performance computing was available only for big companies, as data centers required a large physical infrastructure and have high costs for maintenance (ARMBRUST et al., 2009).

The regular way of building a web application was to use a client-server approach, where the server was a powerful (and expensive) machine. At the same time, new players, such as Google or Yahoo, were rising with bigger missions: *"to organize the world's information and make it universally accessible and useful"* (SPECTOR; NORVIG; PETROV, 2012). The popularization of the internet use incentivized new ways of commerce exchange, yielding an explosion in the amount of data produced and exchanged. It was *just* impossible to store the petabytes of daily-generated data in a single server.

From this point on, the community realized the economical convenience of building and maintaining several low-performance servers, instead of a single high-performance one, even if this this requires a change of culture in the administration of the new datacentres. The new approach (scale-out) is also incompatible with the traditional way of building applications (scale-up), that usually were designed to work on a single server and database. Both approaches are represented on Figure 3.

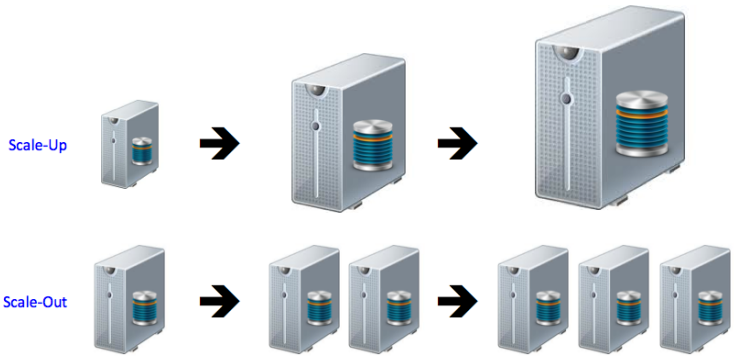

Figura 3: Scale Out vs Scale Up.(DHANDALA, 2015)

Several research initiatives were conducted in this area and a common solution was rising: to distribute data storage and processing. Google, Yahoo and other big IT players helped to build open source tools to make this approach possible, like Hadoop (SHVACHKO et al., 2010).

2.4 Data Integration, NoSQL Movement & Polyglot Persistence

Along with the NoSQL (Not only SQL) movement and expansion of Social Networks, new concepts for Database Models became popular, like Document Store, Search Engines, Key-Value store, Wide Column Store, Multi-Model and Graph DBMS (RANKINGCHART, 2015).

New ways to store and retrieve data were in high demand, as Figure 4 suggests. This chart shows trend of the DB categories' popularity. As (RANKINGCHART, 2015) explains: *In order to allow comparisons, the initial value is normalized to 100. For most of the categories the trend starts with January 2013 but search engines and multivalue DBMS are only collected since February 2013 and May 2013.*

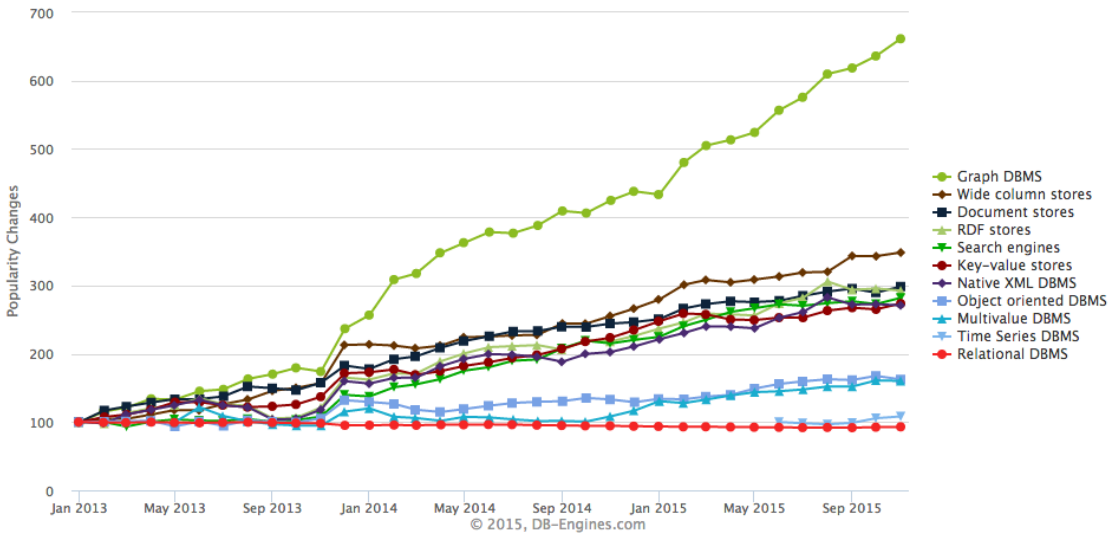

Figura 4: Database Popularity Chart (RANKINGCHART, 2015).

Today, instead of having a single Relational Database Management System (DBMS) for the whole application, it is efficient and cost-effective to have several Data Base Engines, one for each type of data that the application handles. This concept is called *Polyglot Persistence* (SADALAGE; FOWLER, 2012). In (RANKING, 2014) a ranking of the most popular DB engines is presented.

As (SOLAR, 2014) illustrates, polyglot persistence is very useful in the context of e-commerce applications that deal with a catalog, user access logs, financial information, shopping carts and purchase transactions, for example.

The notion of polyglot persistence is built upon the observation that the *nature* of each

data type is significantly different (i.e: user logs imply high volume of writes on multiple nodes, shopping carts need high availability and user sessions require rapid access for reads and writes).

As computing services started to decentralize, developers started to build applications that depended of several data-sources. By this time the use of Web Services and Service Oriented Architecture (SOA) became more popular (ARMBRUST et al., 2009).

2.5 Transitioning Processes

In 1965 Gordon E. Moore, Intel's co-founder, published a paper stating that the number of components in integrated circuits had doubled every two years, and would continue to do so for the at least another decade (MOORE, 1998). Today, this statement is known as "Moore's Law".

A similar trend is stablished for commercial software. Wirth's law, Gates' law (Microsoft) or Page's law (Google) state that "the speed of software halves every 18 months", compensating Moore's law. (WIRTH, 1995)(BRIN, 2009)

In other words, software components evolve as well as hardware evolves. Useful software are usually versioned, updated and patched on a regular basis. As stated by (GLASS, 2001), maintaining software products is not cheap, and generally consumes on average 60% of software costs.

Time-to-market (TTM) is the length of time that takes for from a product being conceived until its available for sale. Minimum-Viable-Product (MVP) is the product with the highest return on investment versus risk (BLANK, 2013). Building a MVP and selling it should be the primary goal of a startup (BLANK, 2013).

The current economy demands faster shipping of software products in order to meet the desired TTM of software products and to deploy MVPs in less time. Cloud computing and Agile Methods made software development and deployment faster, cheaper and easier for a number of companies.

One of the points of the 12 principles of the Agile Manifesto (FOWLER; HIGHSMITH, 2001) is *"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale"*.

This need for speed on software delivery has its downsides, however. Decisions must be made by the IT department on a short time schedule, sometimes leaving not enough time

to make the best choices on the technologies that should be used on a software product. This and other factors oftenly leads to a software migration, replacement or transitioning scenario. In this scenario (part of) a software is replaced with a more suitable alternative.

In the context of Service-Oriented Architecture this might be seen as the replacement of a Service. In a Multitier architecture scenario this might be seen as the replacement of an entire layer. In fact, basically any module of a **modular** software can be replaced, migrated or upgraded.

Examples of Software migrations are numerous on the industry. To number a few:

- Spotify migrated their user base from Postgres to Cassandra(ALBINSSON BARKAS, 2015)
- Google moved from MySQL to MariaDB(CLARK, 2013)
- Twitter moved from Ruby-On-Rails to Java(GADE, 2011)

Our work focuses specifically on Database Transitioning scenarios. We propose a set of guidelines to justify and guide the transitions from RDBMs to NoSQL databases;

2.6 Systematic Mappings

According to (PETERSEN et al., 2008), "*A software engineering systematic map is a defined method to build a classification scheme and structure a software engineering field of interest.*" Systematic Mapping studies provide a global view of a given research field and identify the quantity, results, and the kinds of researches in this field.

A Systematic mapping study is composed by a number of steps (Figure 5).

On the first step, *Definition of Research question*, the questions that must be answered on the survey are defined. On the *Review Scope* step, researchers target the papers/journal sources that will be taken into consideration on the systematic map. After that, the *Search* step is done using a set of predefined search engines and a body of papers (*All papers*) is retrieved.

After an initial *Screening of the papers*, the *Relevant papers* are chosen according to inclusion and exclusion criteria defined by the research team. At this point, the papers that will participate of the study are selected. The selection is based on the title, abstracts and keywords of each paper (*Keywording using Abstracts*).

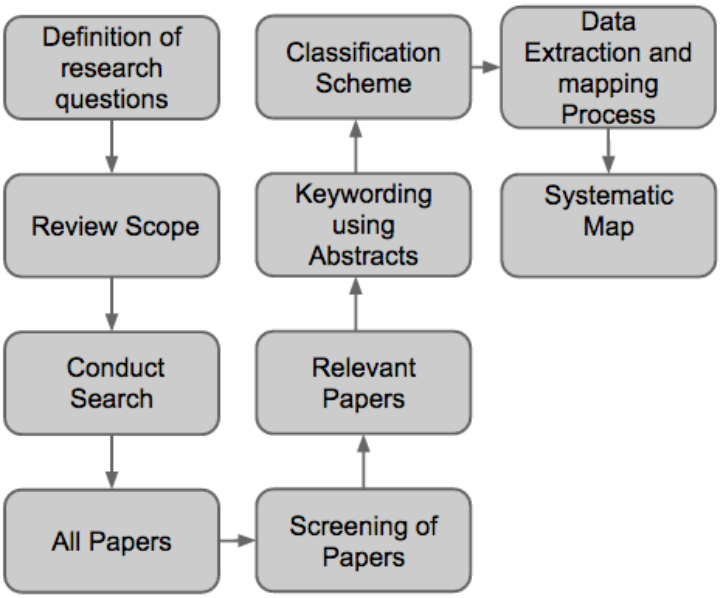

Figura 5: Systematic Mapping Steps (PETERSEN et al., 2008).

After that, a *Classification Scheme* is built, defining different points-of-view (or facets) from which the body of papers will be classified. After matching each paper with the classification schema (*Data Extraction and Mapping Process*), the systematic mapping is performed. In this phase the relationships between the collected data (in the light of the classification scheme) are used to answer the research questions.

In (LEAL; MUSICANTE, 2015) a Systematic mapping study was developed to investigate the use of Service-Level-Agreements (SLAs) on database-transitioning scenarios and to verify how SLAs can be used in this processes. The results of this study are presented in Section 2.8.

2.7 Service Level Agreements (SLAs)

In the field of Law, a contract (or agreement) is generally a written document concerning any point of interest between two parties, each of whom intends to create legal obligations between them. In business environments, contracts are highly necessary to avoid unpleasant situations between a provider and a consumer.

In the context of service provisioning, it is common to refer to contracts (or agreements) as "Service-Level agreements" (SLA's), as the name suggests.

According to *ITILv3's* official glossary (AXELOS, 2012), a Service Level Agreement (SLA) is "*an agreement between an IT service provider and a customer. A service le-*

vel agreement describes the IT service, documents service level targets, and specifies the responsibilities of the IT service provider and the customer."

The agreement consists on a set of measurable constraints that a service provider must guarantee to its customers. In practical terms, it is a document that a service provider delivers to its consumers with minimum quality of service (QoS) metrics. If the service is delivered with a lower QoS than is promised on the SLA, consumers may be refunded or earn benefits that were accorded beforehand.

In the next subsection we present the life-cycle of an SLA, explaining in details each step of it.

2.7.1 The Life-Cycle of an SLA

The Life-cycle of an SLA consists on six steps, as presented on Figure 6. Each step is explained in the following subsections.

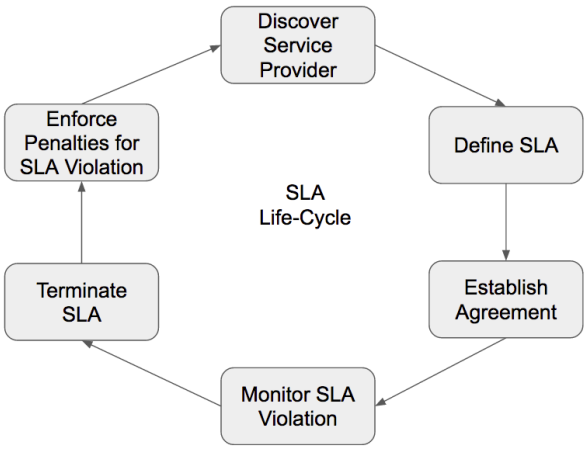

Figura 6: SLA Life-cycle (WU; BUYYA, 2012)

.

2.7.1.1 Discover Service Provider

On the first step *(Discover Service Provider)*, a consumer tipically discovers several service providers that may fulfill his needs on a specific scenario. On the domain of Web Services, *Universal Description, Discovery, and Integration* (UDDI) can be used to retrieve a list of web services that might be used to achieve a goal, for example.

In a Cloud Computing scenario, it is possible to buy computing services from a number of providers, and a cloud-service broker could be used to retrieve a list of available providers. This scenario is very common in cloud computing auctions(SHI et al., 2015), a new concept that arised with cloud computing and was made possible by companies like Hetzner.com, a company that hosts regular auctions of its computing resources.

2.7.1.2 Define SLA

The second step of the SLA life-cycle*(To Define SLA)* is when an SLA is proposed by the provider. Several works, such as (KOUKI et al., 2014) and (KOUKI; LEDOUX, 2012) address the issue of specifying an SLA. WS-Agreement (ANDRIEUX et al., 2005) and WSLA (NEPAL; ZIC; CHEN, 2008) are very important works on this area. As it is presented on (LEAL; MUSICANTE, 2015) , there are several ways to specify an SLA, but none of them are considered a de-facto standard.

(LEAL; MUSICANTE, 2015) also shows that there are several ways to represent an SLA. To enumarate a few, it is possible to represent an SLA as a *i) a natural-language document*, and *ii) an ontology automated test suite (unit tests, integration tests)*.

2.7.1.3 Establish Agreement

As stated on the beginning of this secttion, an SLA is a contract between a provider and a consumer. The points of interest of this contract are known as *Service-Level Objectives* (SLOs).

According to (STURM; MORRIS; JANDER, 2000), SLOs must respect a series of features. They must be *Attainable*, *Repeatable*, *Measurable*, *Understandable*, *Meaningful*, *Controllable*, *Affordable* and *Mutually acceptable*.

Each SLA may explicitly define the "hardness" of its SLOs in terms of time. In other words, an SLA can specify what is the fault tolerance rate of each of its SLOs.

An example can be given to clarify this concept: Two telecommunication companies may have different SLAs regarding the uptime of their services given a fixed period of time. **Company A** may assure that their uptime is 99.9% of the time under the period of a year, and **Company B** may assure that their uptime is 99.99%.

In this scenario, it is said that the SLA of Company B is "harder" than the SLA of Company A. Figure 7 compares the Availability SLO defined in SLAs of some popular

cloud services.

Service Level Objective: Availability

Compose.io

Promised SLA	Penalty
< 99.98%	Discount: 20%

Locaweb.com

Promised SLA	Penalty
99,0% to 99,4%	Discount: 5%
95,0% to 98,9%	Discount: 10%
90,0% to 94,9%	Discount: 20%
< 89.9%	Discount: 30%

Amazon RDS

Promised SLA	Penalty
99,95% to 99,0%	Discount: 10%
< 99,0%	Discount: 20%

Figura 7: Service Level Objective: Availability on Cloud Services (LOCAWEBSLA, 2015)(COMPOSE.IO, 2015)(AMAZONRDS, 2015)

After chosing the service and defining the SLOs, an SLA might be established between a consumer and the provider - the third step of th SLA life-cycle.

2.7.1.4 Monitor SLA violation

The fourth step *(Monitor SLA violation)* is one of the most important on the SLA life-cycle. It is in this step where the consumer protects himself against unfulfilled SLOs.

Several frameworks and processes might be used to measure and monitor SLA violation. As SLAs do not present a standard way for representation, it is also difficult to find a standardized way to monitor SLA violation.

(RANA et al., 2008) presents three “monitoring-models"for SLA violation:

- *All-or-nothing*: In a all-or-nothing situation, **all** the SLOs must be satisfied to guarantee that the SLA is still valid.

- *Partial*: In a partial scenario, the SLA specifies what are the SLOs that may not be broken, and declares rules less important SLOs.
- *Wheighted Partial*: In a wheighted partial scenario, the consumer and provider specify thresholds that must be met for specific SLOs. If the service quality drops below a threshold, a penalty can be applied.

In the context of Web Apps it is also possible to monitor SLA violation with the help of web applications, such as (DATADOG, 2014), (APPSEE, 2014) and (NEWRELIC, 2014).

2.7.1.5 Terminate SLA

An important question of the life-cycle of a SLA (fith step) is *when* the termination of an SLA is needed. If the termination is due to an SLA violation, it is important to know what party broke the SLA and what are the consequences for the parts.

These situations must be explicitly declared in the initial SLA to avoid unpleasant situations between the provider and the consumer in the future.

2.7.1.6 Enforce Penalties for SLA violation

The sixth step *(Enforce penalties for SLA violation)* can be performed if the penalties that were previously defined are being hit in a regular basis or if both parties of the SLA agree. Several works, as (LEE et al., 2010) propose penalty models/processes for SLA violations. These processes can be analyzed and used by the SLA parties.

2.8 Our Systematic Mapping

To provide us a better understanding over the use of SLAs in component / service migrations, we have performed a Systematic Mapping study to assess the use of SLAs in database transition scenarios, specifically on migrations from relational databases with NoSQL ones.

This study is available on Appendix A.

2.8.1 Identified problems

As a result, we have analyzed over 70 publications closely related to the use of SLAs in migration scenarios. The study revealed a number of interesting outcomes, and we emphasize two of them below:

- No publication was found addressing the problem of measuring the overall improvements after a database transition. Several benchmarking frameworks, such as TPC-H, TPC-DS and YCSB were identified(MOUSSA, 2013) during our survey, though. These benchmarking frameworks could be a good starting point to develop new tools and specialized frameworks to solve this problem and might be used in our study to validate that a migration was successful.

- (SAKR; LIU, 2012), (ZHAO; SAKR; LIU, 2013), (KLEMS; BERMBACH; WEINERT, 2012) and (XIONG et al., 2011) propose SLA-centric/User-Centric solutions to monitor the performance of web applications. All these solutions are technology-agnostic and could be used to monitor the performance improvements promised by a database transitioning process. Industry experts also pointed out that there are some services, such as New Relic (NEWRELIC, 2014), Appsee (APPSEE, 2014) and Datadog (DATADOG, 2014) that provide SLA-monitoring tools for web apps.

  The systematic mapping revealed no open source solution to monitor Application SLAs in a user-centered view (application level).

3 The Problem

In this section we detail the problem that we analyze in this study. A good understanding of the previous sections and of the systematic mapping available in (LEAL; MUSICANTE, 2015) is required.

3.1 Problem Breakdown

For a long time the industry developed applications storing data on relational databases. Graph databases, Search Engines, Document stores, Wide Column Store and other types of database systems emerged over the last years as a demand of the industry.

New categories and types of database systems are capable of processing and accessing data in new ways that relational DBs do not support. In other words, for some application, using relational databases may not be the best fit.

When developing an application, it is not always possible to use the technology that is most suited to the application use cases and requirements. This situation is due to a number of reasons. In an Agile project, for example, not always the team knows all use-cases and application requirements at once; as the Agile Manifesto states: "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage." (FOWLER; HIGHSMITH, 2001).

Furthermore, using several databases on the early releases of a software product may add unnecessary complexity to the code base and might become an overengineering problem over time.

The fact that some applications are built not knowing beforehand all the requirements, use-cases and expected workload on production environments - *(**how much** will it scale?)* - often leads to database transitioning scenarios.

Applications that work with more than one database type are popular at the present

time, as (SADALAGE; FOWLER, 2012) reveals. Some companies, as Instaclustr (INSTACLUSTR, 2014) and (ELASTIC, 2015), emerged in order to provide support and consultancy to database transitioning scenarios.

There are other cases, also, where an application is developed using an existing database technology, as MySQL, and a number of improvements are made on it, leading to the rise of new databases. Twitter FlockDB (FLOCKDB, 2014), a distributed & fault-tolerant graph database, emerged in these conditions.

3.2 Database transitions in the industry

Database transitioning processes are not easy and straightforward tasks, depending on the size of the application. A number of steps is required when performing database transitions, as revealed on Section 1.

On the other side, a structured process is not always followed when switching databases on production environments. (LEAL; MUSICANTE, 2015) revealed, for instance, that database transition scenarios on the industry follow non-standardized methods and that they may differ significantly among applications. This situation may lead to losses, rework and a hard to maintain codebase.

There are some industry reports, as the ones from Coursera, a leading Education Technology company with over 10 million users (WIKIPEDIA-COURSERA, 2014), where more than one database transitions were made in a brief period of time.

Frank Chen, member of Coursera's engineering team, enlightened the reasons why Coursera replaced part of its database technology from MongoDB (NoSQL DB) to MySQL (Relational DB) (HAO, 2014b) .

In other post (HAO, 2014c), Chen also reveals that part of the new MySQL architecture was migrated in a second step to Cassandra (NoSQL DB). A blog post from Coursera's Engineering team confirms and discusses more about the transition from MySQL to Cassandra (COURSERATECH, 2014).

Analyzing all the conditions evidenced on this section, a problem to our study was defined: How can a database transition be made in a pragmatic manner? What are the steps and pitfalls to be avoided in relational to NoSQL transitions?

3.3 Proposed solution

When an application or website grows, a good strategy is to isolate the application server and database server in two distinct machines, as (ELLINGWOOD, 2014) suggests. If a machine shares its resources among Web Servers, DB Servers and other applications, for example, a high CPU load on the web server side may impact the performance of the Database, downgrading its performance.

In this work we propose a set of steps to guide the transition from relational databases to NoSQL ones. The suggested guidelines make use of a set of Service Level Agreements (SLAs) to guide the whole process. These guidelines are represented on Figure 8 and will be discussed along this chapter.

The guidelines proposed in this work assume that the database server is isolated on its own machine.

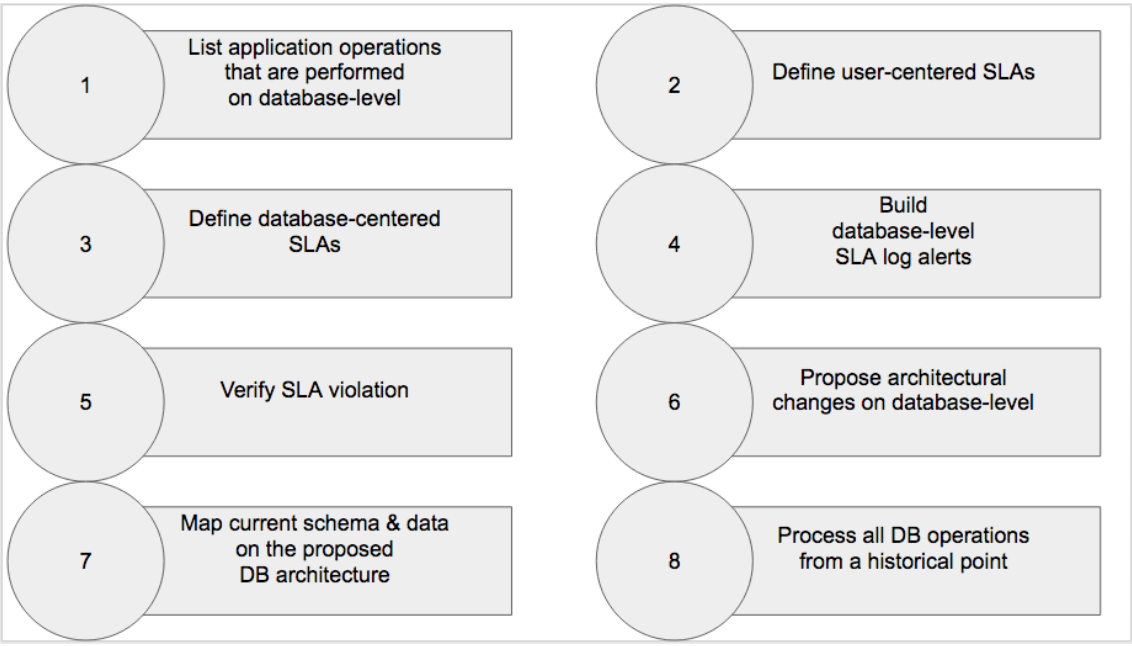

Figura 8: Relational to NoSQL Steps.

3.3.1 List application operations that are performed on database-level

A database transition can be motivated by a number of reasons, as budget, lack of support from the community and performance issues. Migrations scenarios motivated by performance issues are the focus of this study, as other reasons might be very specific to each application / transition scenario.

Performance issues might be caused by database growth or by new application requirements. The first step in a transition scenario is, then, to identify *which* are the application operations or requirements that are driving the transition.

To identify these operations, it is necessary to explicitly enumerate the application operations that are performed on database level. To illustrate the process of enumerating these operations, consider the following application examples:

A retail business-intelligence application: A retail business-intelligence application can be used by large retail corporations and supermarket chains. Some use-cases that perform database-level operations on this kind of applications may be:

- To process consumer purchases;
- To clusterize customers by their consumption profiles;
- To export summarized reports of transactions that happened last week;

Social Networks: A social network is a category of applications where users generally may befriend / follow other users, publish posts and share their own content. On this kind of applications. Some use-cases that perform database-level operations are:

- Follow or befriend another user;
- Publis posts;
- List user timeline;

3.3.2 Define user-centered SLAs

For each database operation enumerated on the previous step, a **user-centered** wheighted-partial SLA is defined with the stakeholders of the application. For these SLAs, two thresholds must be explicitly defined for each operation: an **"ideal threshold"** and a **"tolerable threshold"**.

Ideal threshold is the performance level that is expected by application users. The ***tolerable threshold*** is a threshold where users can still use the application, but the user experience with the application is downgraded.

When the ***tolerable threshold*** is broken, user experience is dramatically affected and (part of) the application may not be operational for its users.

An example of user-centered SLA is given in the context of the retail business-intelligence application previously defined to illustrate this concept:

- **Process consumer purchase (Store credit card transaction on my Data Warehouse**

  Ideal Threshold: up to 1 seconds;

  Tolerable threshold: up to 1 minute;

  In other words, the user expects a purchase to be processed in one second. If it takes up to 60 seconds, the application can still be used, despite it may affect user's experience. If a purchase takes more than 60 seconds to be processed, users may get really frustrated and long lines may be expected on the cashier.

- **Export summarized report of transactions that happened last week (Retrieve and perform aggregation operations on selected records)**

  Ideal Threshold: up to 24 hours;

  Tolerable threshold: up to 72 hours;

In other words, the app user expects that a summarized report of last week's transactions is made available in up to 24 hours. If the exported report takes more than 3 days (72h) to be generated, the application is not useful to the user anymore, as the business strategy might be severely affected by application performance.

The ***SLA Delta*** between the ***Ideal Threshold*** and ***Tolerable Threshold*** is defined as the number of ***Ideal SLAs*** that can fit inside ***Tolerable SLA***. From this concept, we can define the ***SLA Delta*** for the operations defined above:

- **Process consumer purchase (Store credit card transaction on my Data Warehouse)**

  Ideal Threshold: up to 1 seconds;

  Tolerable threshold: up to 1 minute;

  SLA Delta: 6.000% (60x)

- **Export summarized report of transactions that happened last week (Retrieve and perform aggregation operations on selected records)**

  Ideal Threshold: up to 24 hours;

Tolerable threshold: up to 72 hours;

SLA Delta: 300% (3x)

The **SLA delta** is always equals to or greater than 100% by definition, and will be used as a parameter on the following steps of the proposed guidelines.

As stated previously, the **ideal threshold** and **tolerable threshold** should be defined with the application stakeholders as quantifiable metrics. The **SLA Delta** can be obtained as the division: **tolerable threshold** / **ideal threshold**.

As we will be targeting performance issues at database level, **ideal threshold**, **tolerable threshold** and **SLA delta** can generally be seen as a time-related metrics, as it is shown on the examples above.

In fact, *time* is the main metric when benchmarking databases, as presented on (ENDPOINT, 2015) - operations/sec and latency. Other quantifiable metrics and Key Performance Indicators (KPIs), as *processed workload* and *client CPU / memory usage* can also be used as metrics to the thresholds, however. I.E: it is also possible to define thresholds related to the client CPU usage, for example.

On Figure 9 it is possible to graphically visualize the concepts of **Ideal Threshold**, **Tolerable Threshold** and **SLA Delta**.

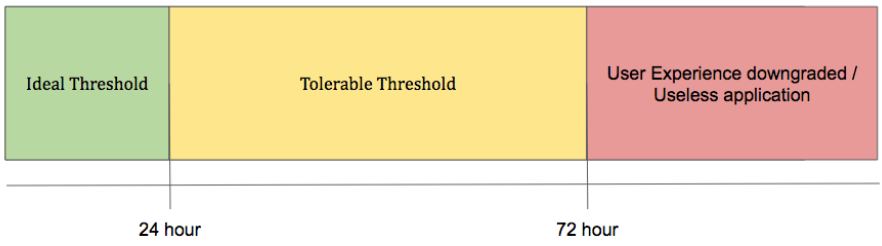

Figura 9: SLA Thresholds - 3x SLA Delta.

3.3.3 Define *database-centered-SLAs*

On a multitier architecture, databases are generally the last layer to be reached on a user operation. Several other steps are necessary to process a user request, as field validations, security filters and business logic.

From the user-centered SLAs defined on the previous step, another set of SLAs should

be proposed on this step, now on database level. These SLAs are called ***database-centered-SLAs*** in our guidelines and are proposed from the ***user-centered SLAs***.

By definition, ***database-centered-SLAs*** should be "harder" than the user-centered SLAs, as other operations are needed in the process of processing a user request. A ***database-centered SLA*** specifies the Quality-of-Service (QOS) that is expected from the database service while processing an operation.

Database-centered-SLAs should only be defined for the operations that perform transactions at database-level. The same concepts that were defined for ***user-centered SLAs (Ideal Threshold, Tolerable Threshold and SLA Delta)*** are valid for ***database-centered SLAs***.

An example of ***database-centered SLA*** is given for the operations that were enumerated on the the previous step:

- **Store credit card transaction on my Data Warehouse** ***(process consumer purchase)***

  Ideal Threshold: up to 0.2 seconds;

  Tolerable threshold: up to 8 seconds;

  SLA Delta: 4.000% (40x)

- **Retrieve and perform aggregation operations on selected records** ***(export summarized report of transactions that happened last week)***

  Ideal Threshold: up to 10 hours;

  Tolerable threshold: up to 20 hours;

  SLA Delta: 200% (2x)

3.3.4 Build database-level SLA log alerts

After defining ***user-centered*** and ***database-centered SLAs***, application architects must define a rate of requests that can be executed within the tolerable-threshold level for each operation.

This ***rate of faulty requests (ROFR)*** is necessary to avoid unnecessary alerts caused by infrastructural instability, inherently present on cloud providers, as Figure 7 presents.

3.3.4.1 Defining a Rate of Faulty Requests (ROFR)

The ***rate of faulty requests*** is used in our guidelines to alert application developers that some operations are not being executed with the desired QOS.

With a defined *ROFR*, it is possible to build a log / application analyzer that will track if any database operations are breaking the tolerable threshold, or if the rate of faulty requests is above expected.

If the ***ROFR*** exceeds the expected level or if a request exceeds the tolerable threshold, an alert should be sent to the Database administrators and maintainers of the application, indicating that an SLA violation was found and that further analysis is needed. This alert can be sent using regular alerting systems, as Email, SMS and push notifications.

Log analyzers and alerts can be implemented within the source code of the application or using external services, such as New Relic, Papertrail and Logstash.

Once an alert is sent, it should contain the timestamp of the moment when the SLA violation was detected and even the process list of what was being executed on the database at that time.

It is possible to define a rate of faulty requests for the examples given above:

- **Store credit card transaction on my Data Warehouse**

  Ideal Threshold: up to 0.2 seconds;

  Tolerable threshold: up to 8 seconds;

  SLA Delta: 4.000% (40x)

  ROFR: 10%

- **Retrieve and perform aggregation operations on selected records**

  Ideal Threshold: up to 10 hours;

  Tolerable threshold: up to 20 hours;

  SLA Delta: 200% (2x)

  ROFR: 30%

In other words, if more than 10% of the "Store credit card transactions on my Warehouse" are performed on a **Tolerable threshold** level, an alert is fired. If any transaction of this kind lasts more than 8 seconds, another alert is also fired.

In the second case, if up to 30% of the transactions are processed within the tolerable threshold, no alert is fired. If any transaction lasts more than 20 hours to be executed, one alert is fired.

3.3.5 Verify SLA violation

When an SLA violation alert is received, it is necessary to know what caused it.

Failures might be possible at machine-level or on the infrastructure that hosts the database server. Hardware issues, network problems and cyber attacks are some factors that may downgrade database performance.

Extensive and detailed work from application developers might be needed to detect the real cause behind an SLA violation, and the source of each violation is very singular for each application.

Another possible point of failure is the application itself. The mean number of operations per second may have increased, resulting in a downgraded performance of the database; A new feature might be demanding more DB resources and bugs might end up performing faulty requests on the database.

3.3.5.1 Further Analysis

SLA violation alerts contain the exact timestamp of when the violation ocurred, as well as the database process list that was active on the moment of the violation.

Popular relational databases, as MySQL, Oracle and Postgresql implement a feature called *point-in-time recovery* (MYSQLRECOVERY, 2015) (ORACLERECOVERY, 2015) (POSTGRESRECOVERY, 2015). This feature allows a DB to be dumped and restored to a specific point in time.

As the alert contains the timestamp of when the SLA violation was triggered, it is possible to clone the relational database and restore it to the exact time before the SLA violation was triggered. In this cloned environment it is possible to investigate in detail what caused the SLA Violation.

If everything is working properly (no issues were found on the database server and on the application), two actions are possible: relax SLA thresholds or propose changes on the current DB Architecture;

3.3.6 Propose architectural changes on database-level

Not always it is necessary to replace the database to address a performance problem. SQL tunning, denormalizing tables and creating indexes are some ways to improve the performance of applications that use relational databases.

Another option to address performance problems on relational databases would be to scale-up the current DB, buying more powerful hardware. This scenario is not covered by this work, as budget is always a finite resource on companies.

If the SLA remains broken after the architectural changes have been performed on the current DB infrastructure, a NoSQL strategy might be recommended. In this case, a new Database Model (Graph DBs, Document Stores, Key-value stores, etc.) and technology (Neo4j, MongoDB, Couchbase) might be chosen.

Several works, as (HAN et al., 2011) and (LEAVITT, 2010) present overviews of NoSQL databases. A good starting point to choose which category of NoSQL Database is the best fit for an application is the CAP theorem (SADALAGE, 2014), presented on Figure 10.

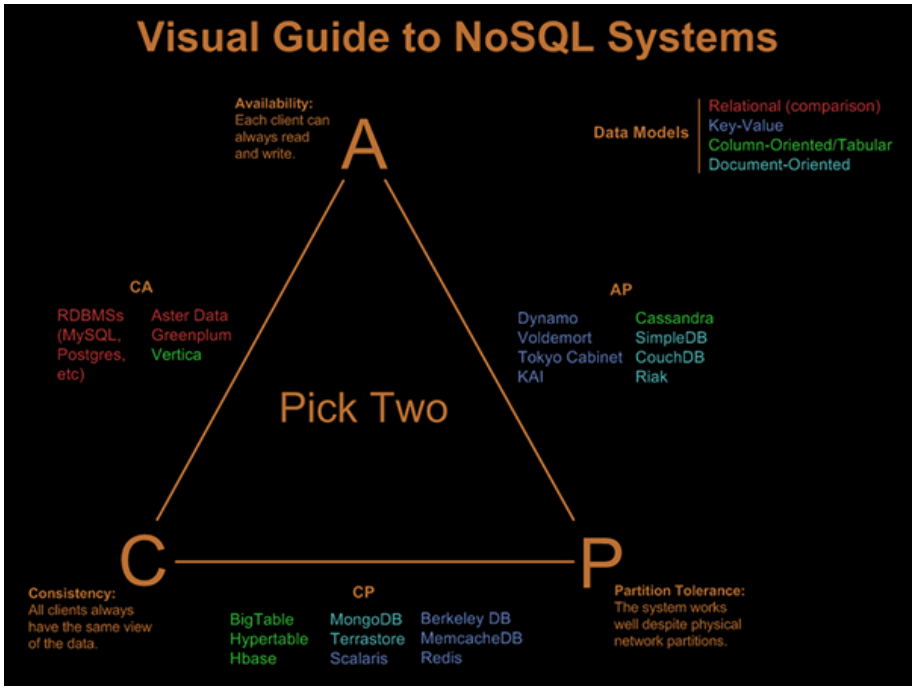

Figura 10: CAP Theorem.(HAO, 2014a)

The CAP Theorem states that that it is possible to have two from the three capabilities in a database:

- Consistency (all nodes see the same data at any point of time);

- Availability (a guarantee that every request receives a response about whether it was successful or failed)
- Partition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system)

Relational databases have consistency and availability, resulting in a scale up strategy, presented on Figure 3.

3.3.7 Map current schema & data on the proposed DB architecture

After choosing the new NoSQL database, it is necessary to map the table's rows and relationships into the concepts of the chosen NoSQL technology. Despite the fact that some NoSQL DBs are schemaless, defining a schema and building indexes help to leverage the perfomance of NoSQL DBs.

To compare the performance of RDBMS vs NoSQL on a specific scenario, the same data should be present on both databases. A Dump & Restore procedure should be done at this point. i.e: The data should be dumped from the RDBMS and imported on the proposed NoSQL architecture.

The process of mapping (part of) the relational database entities into the new NoSQL schema is very unique to each application and the chosen NoSQL technology. This is a wide topic and not a subject of this work. (BAHL, 2014) shows, for example, how the models of an application might be mapped between different relational & NoSQL technologies.

Once the new DB architecture is restored with the production data, it is possible to compare the results of the proposed architecture and the old architecture, which is done on the next step.

Over the last sections we have defined thresholds for operations on a Business Intelligence application. The relational database that stores application data on this kind application should have several entities, as users, products and commercial transactions. Each of these entities have their own tables, and tables are connected from relationships.

On a NoSQL architecture, all these entities and relationships could be combined on a single JSON document, as presented on listing 3.1.

Listing 3.1: BI application commercial transaction represented as a single document.

```
{
"id": 12089367123
"user": 12908376123,
"items": [{"id": 01,"category":"food","name":"rice"}, {"id": 21,
    "category":"drinks","name":"soda"}]
}
```

3.3.8 Process all DB operations from a historical point

In this step it is possible to compare the performance of the relational database and the proposed NoSQL architecture. From this point, the following elements should exist as outcomes from the previous steps:

- A production relational database;
- A clone from relational database;
- The proposed NoSQL technology & data model;
- The same data should be available on the cloned database and on the proposed NoSQL database;
- Logs of the relational database;
- One or more SLA violations;

In a scenario where the relational database will be completely replaced by a NoSQL alternative, it should be possible to execute the same operations on the relational DB and on the proposed NoSQL alternative in any given point in time.

Some DB transitioning processes, however, are performed using a *polyglot persistance* strategy, where an application has several databases. In this case, only some of the application entities will be present on the NoSQL model, and the application data layer will be composed by two or more databases. In this case, only a subset of operations can be executed on the NoSQL side.

A question that may arise at this point is *"What should be the starting-point to compare requests between the relational and NoSQL databases looking for SLA violations?"*

The most complete strategy would be to restore relational database to the oldest point in time where the logs enable and to perform all operations from that point to the point where the SLA violation was triggered. By doing so, it is possible to eventually discover SLA violations that do not exist in the relational architecture and that may arise on the NoSQL architecture.

There are some cases, however, where the volume of data transactions is too large to go back to the oldest point in history. In this cases, a good strategy could be to retrieve all processes that were executing immediately before the SLA violation was fired and go to the point in time when the first of this processes started.

From the examples given on the previous sections, an alarm would be triggered when the database lasts more than 20 hours to generate a report. When this situation happens, the admins should check the meaningful operations that were running immediately before the alarm is triggered and start to execute the requests in a chronological sequence.

3.3.9 Transitioning Recommendations

If the proposed NoSQL architecture is able to handle production operations without triggering SLA violations, this is a good sign that the NoSQL might be able to handle the production load.

Transitioning a production database is not an easy and straightforward task, however. It is necessary to change the source code of applications, eventually change some tests and some bugs may be arise during the process.

Software engineers and database dministrators should assess the impacts of changing database entities and build a transition roadmap considering the following steps:

1. Choose entities to be transitioned: as stated previously, it is not necessary to change all database entities in once. If SLA violations are being triggered only for some entities, replacing these entities in a NoSQL architecture can solve the problem.

2. Change source code, tests and documentation: Changing the database results in changes on the source code of application and eventually changes on tests and documentation. This should be done as a second step in a transition process.

3. Parallelize production calls to both databases: A good practice on database transitioning is to run both storage solutions during a period of time. This means that

the relational database is still used as a master storage and the NoSQL is used as a secondary storage. Doing so, a production request should be handled in parallel, but the results of the operations on the secondary database are ignored.

4. Perform integrity checks and SLA verifications: At this point it is possible to check if the production requests are correctly being handled by the NoSQL DB. No further SLA violations should be fired at this point.

5. Change for a small user base: A small user base can be used to make sure that no other issues will arise from the database transition. This users will have the results from the secondary (NoSQL) database as a result.

6. Change for all users: After a small user base is tested and no SLA violations were found, the transition can be performed for all production users. In this point, (part of) the relational database can be deprecated.

If the engineers decide not to transition the database in once, it is possible to perform these steps in a cycled manner. i.e: it is possible to transition the entities where SLA violations are being intensively fired at first and then perform other transitioning cycles. This way, the negative impacts of a database transition is amortized.

4 Validation

To validate our guidelines, we searched for open-source tools that could possibly require database transitions. As stated on section 1, a database migration is justified when the alternatives have better performance/manutenability than the classic RDBMSs and/or the cost to have a similar performance on the relational database architecture is significantly higher.

This condition (assessing the need for a database transition) is, then, only verifiable on production or simulated environments. Wordpress(WORDPRESS, 2015), world's most popular Content Management System (CMS) (BUILTWITH.COM, 2015), can be used to illustrate this problem.

Wordpress is built on the top of relational databases, such as PostgreSQL and MySQL. So, to justify a database transition on a wordpress-based application, we must *(i)* have access to a deployed and active version of the CMS or *(ii)* build a simulated environment and load it with posts, pages, comments and other entities that are present on a regular wordpress website.

Having access to the production version of a heavily used tool, such as Wordpress, is not easy, as the owner of the CMS must give access to sensible information of its database, users and posts.

Other open source tools were considered on this phase of the research, such as Redmine (REDMINE, 2015) and Moodle (MOODLE, 2015), but the same problem that we had trying to accessing sensible information of Wordpress environments was found on these tools.

Besides that, famous open source tools usually have a large user-base. This results in a large support from the community towards the development of the software. With that, if a database transition from RDBMs to NoSQL is needed, it possibly have plugins/addons from the community, as (PARIS, 2015).

Facing this access restrictions to popular open-source projects, to validate the Guide-

lines proposed on the previous chapter, we have built scenarios that can be easily mapped to real-world applications.

The application proposed on section 4.1 is used to illustrate our case studies.

4.1 Assessing database transition on a social media monitoring application

To validate the guidelines presented on Chapter 3 we have modeled *the core database schema* of a social media monitoring application. Some examples of industry platforms of this kind are (SPROUTSOCIAL, 2015), (MENTION, 2015) and (BUZZMONITOR, 2015).

This category of applications are capable of searching and storing posts from social media platforms, as Twitter and Facebook. On the setup of a new account, users usually define a boolean query of the terms that they want to monitor and the platform begins to monitor *Application Program Interfaces* (APIs) from social networks, searching publications that match the boolean query.

4.1.1 Application Requirements and use cases

From a user perspective, this kind of application might be used to search for relevant topics across social media posts, to be aware of public opinion about brands (brand awareness) or to assess market for a product, for example.

From a software engineering perspective, some use-cases of this application could be:

1. **Create, retrieve, update and delete (CRUD) user account**
2. **CRUD users' terms to monitor**
3. **Retrieve posts by ids** - Given a set of post ids, the user is able to visualize them on the user interface (UI).
4. **Classify posts (add/tags tags)** - N tags (subjects) can be added or removed from a set of posts.
5. **Filter captured posts by filters** - Some possible filters are:
    - Boolean Query: Search posts that mention a boolean query string. *i.e: (Brazil AND Neymar) OR (Orlando AND Kaka)*

- Date: Search posts that were published within a date range. For example: posts from 2015/01/01 to 2015/02/01.
- Tags: Search posts that match a tag query. i.e: posts about "Soccer Player" and "High Salary".

Other possible features and use-cases of this kind of applications are not listed to ease the understanding of the scenarios that are presented on the following sections.

4.1.2 Application Architecture

Building applications like this one brings some challenges and decisions to software architects and engineers. As the scope of the project grows, it can be a wise decision to split the application in various sub-applications (or components).

Moreover, it is quite costly to manage the whole application from a single code base. The source code needs to handle users registration, query several API's and automatically analyze the sentiment of posts using Natural Language Processing techniques, for example.

Each of these components may have several other sub-components, or microservices. A microservice architecture brings some benefits to the application, such as maintainability, reuse and simplicity of deployment. A microservice that automatically detects the language from a given a string, for example, can be built and used in all components that need to accomplish this task.

Components or service-oriented architectures are useful as it enables using different technologies for distinct purposes. Just as in polyglot persistence, the idea is to use the best technology for each task.

Several sub applications could be proposed and created to work with each social network that is supported by the application. i.e: One service/component might be able to handle all the content related to *Facebook*, other component may be responsible for *Twitter* and another one for *Pinterest*-related features.

On Figure 11 we illustrate a component-based architecture for the proposed application.

Five main components can be extracted from the proposed architecture:

- **MainApp Component**: This component can be seen as a web application where users can create their account, manage their social profiles, edit the terms that they

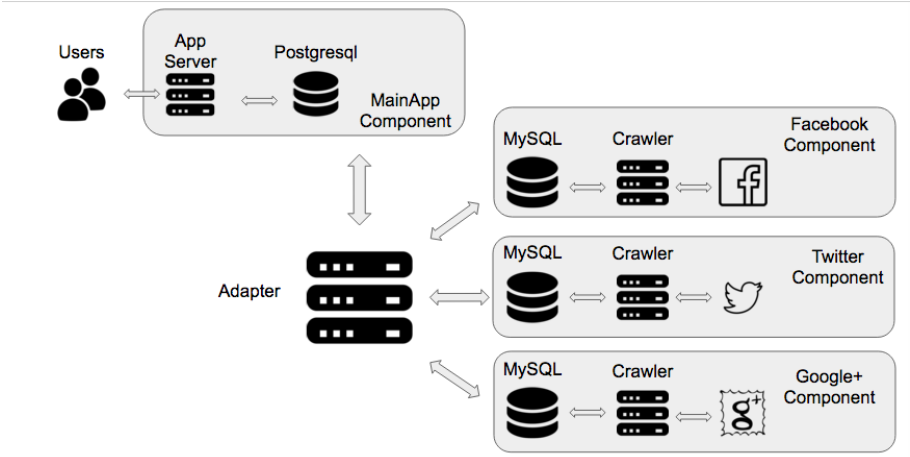

Figura 11: Proposed architecture - Social Media monitoring app.

want to monitor and where the user-interface is presented. A web framework, such as Ruby on Rails or Django can be used to build this component.

- **Twitter, Google Plus and Facebook Components**: These components are responsible to provide the necessary communication between the MainApp component and Social Networks. The strategy to build a component for each social network was assumed as different social networks have distinct APIs, with different entities and distinct communication patterns.

  In other words, *Facebook* may release its API through a HTTP REST perspective, while *Twitter* may release Software Development Kits (SDKs) to a restrict set of programming languages (Gems for Ruby, JARs for Java, for example) and Google plus may release its API through the SOAP protocol, making XML the standard way to handle data on its scenario.

- **Adapter Component**: The adapter component is responsible to integrate and to act as an interface between the different schemas that compose the data layer from other components (Facebook, twitter and Google Plus).

As presented on Figure 11, the architecture of the proposed application relies a Polyglot Persistence strategy. Two distinct databases are used on the app: Postgresql and MySQL. On real-world scenarios, this situation might be caused because of legacy code, by internal decisions of application architects or any other reason.

4.1.3 When application grows, problems may arise

Once the application is released, it is desirable that no performance issues are found. Operations executed by application users should, theoretically, be at a acceptable performance level.

In the Facebook component, for example, it is expectable to have no performance issues with tens to hundreds of rows stored on the MySQL database. However, worldwide events, as the World Cup or national elections, may start to be suddenly monitored, bringing an unusual workload to the components.

In these cases, or when the application starts to keep track of millions or billions of data records, application performance and user experience may be hardly affected if the engineering team have not done sufficient load and stress tests.

Following this scenario, after some months since application release, the number of records on MySQL databases from the three social-networks components may grow exponentially. A simple query that checked if a column contains a substring starts to analyze millions of posts instead of hundreds.

Dependendo dos termos cadastrados, depois de alguns meses, o numero de posts nas bases MySQL dos tres subsistemas pode crescer bastante, e uma simples consulta que verificava se um post continha uma substring pode passar a analisar milhoes de posts ao inves de centenas. Intuition tells us that as the number of posts to be analyzed grows, the query time also grows.

4.1.4 The application needs other databases?

On the scenario described on section 4.1.3, application users may start to complain that the application is taking too long to process their requests. This situation ends up slowing the overall performance of the users, as an action that could be processed within seconds starts taking minutes to be finish.

User complainments about performance, however, may be related to a number of different factors. To number a few: the business logic might be taking too long to execute, the number of records in the database may have grown in an unexpected way, the database architecture might not be the best choice to handle application data and failures on network or server hardware may exist.

Given this scenario, we will focus our analysis on the components that are responsible

for storing the posts on MySQL databases. More precisely, our study will be focused on the Facebook component, to present a clearer definition of the problem.

A possible claim from the application users' is that the application is getting slower month after month to filter, process and present the posts on the user interface (UI).

It is expected that the number of posts grow month after month. Intuition tells us that the search speed slows down as the number of posts in the database grows, and application developers and DBAs may use this intuition as a hint to find start searching for the real cause of the problem.

This way, the engineering team from the Facebook component might have a strong intuition that the cause of the slowness is on the data layer (MySQL). Engineers may also have an intuition that it is a specific kind of SQL query that is taking too long to execute, consuming too much CPU & memory resources. The team may also suggest that a NoSQL technology could be a better fit for the use case that is performing below users' expectation.

In this case, the guidelines proposed on the previous chapter are a good fit, as they are useful to verify if any problems/bottlenecks exist on the database side and to check if the a NoSQL architecture could be a better fit to the scenario.

4.1.5 The application setup: Server

In this section we detail the server technical specifications that were used to host the data layer of the application that we presented on the previous section. We have selected a **T2.SMALL** instance from (EC2, 2015) to host it, as it is a general purpose and low-price instance. T2.SMALL instances feature the following configuration:

- High Frequency Intel Xeon Processors with Turbo up to 3.3GHz Burstable CPU, governed by CPU Credits, and consistent baseline performance
- 1 vCPU
- 12 CPU Credits/hour
- 2 GB RAM

4.1.6 The application setup: Software

The following software configuration was installed on the servers:

- Operating System: Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-48-generic x86_64)
- Secure Shell
- MySQL Version: 14.14 Distrib 5.5.44, for debian-linux-gnu (x86_64) using readline 6.3

4.1.7 The application setup: Data

To retrieve the data that is used on the scenarios, we have developed a web crawler that gathers posts from Facebook and store them on our MySQL database. The data was captured from public posts on popular Fan Pages.

A total number of 3.332.534 posts (more than 3 milion posts) were captured and the Dump file with these posts is available on XXXXXXXXXXX.

4.1.8 The application setup: Database schema

The first database schema was built with the intention of being optimized to production environments, and building **de**-normalized schemas help to leverage application performance, as revealed by (SANDERS; SHIN, 2001).

On Figure 12 it is possible to view all the columns that compose the **Posts** table. All post entities, like *message* (content of the post), *link* and *number of likes* are present on this table.

Figure 12 reveals that "tags" is a text field on the posts table. By internal convention from the engineering team, a tag of a post can be represented by a string with a defined format: *#**username_ tagname**#*.

A "tags" cell might contain several tags. Listing 4.1 gives the representation of a tags cell with multiple tags within:

Listing 4.1: Tag field standard format

```
1 #username1_tag1##username2_tag1##username3_tag1
```

When the user adds a tag to a given post, the "tags" field is concatenated with the given tag. A SQL UPDATE clause example to this situation is given on listing 4.2.

Listing 4.2: Update Tag - SQL

Column	Type
id	bigint(30)
post_id	bigint(20) unsigned
comment_id	bigint(20) unsigned
comment_reply_id	bigint(20) unsigned
fan_page	tinyint(1)
collected_from	enum('POSTS','FAN_PAGE','GROUP')
created_time	datetime
updated_time	datetime
message	text
type	varchar(20)
link	text
name	varchar(250)
caption	varchar(150)
description	text
picture	text
source	text
mood	float(3,2)
icon	varchar(250)
likes	int(11)
comments	int(11)
page_id	bigint(20) unsigned
group_id	bigint(20) unsigned
page_url	varchar(400)
author_id	bigint(20) unsigned
author_name	varchar(150)
author_gender	char(1)
upload	tinyint(1)
in_reply_to	bigint(20)
tags	text
replied_element_id	varchar(70)
replies	smallint(6) unsigned
shares	int(11)
term	text
archived_by_user	text
archived	tinyint(1)
location	varchar(100)

Figura 12: Posts table.

```
UPDATE post set tags=concat(tags, '#username3_tag1#') WHERE id=1;
```

To remove a tag, a "replace" operator can be used, as shown on listing

Listing 4.3: Remove Tag - SQL

```
UPDATE post SET tags= REPLACE(tags, '#username3_tag1#', '') WHERE id=1;
```

The decision to store the tags in a field instead of storing it on another database was taken to avoid JOINS between different tables on the application.

4.2 Using the guidelines

As shown on chapter 3, the guidelines proposed on this work are useful to assess and guide database transitions in production-ready or simulated environments. We may use the guidelines as a good starting point to assess if a database migration to NoSQL is really needed or if the problem can be solved by improving the current DB architecture or if it is not even located on the data layer.

Throughout this section we use the guidelines in a step-by-step strategy, as shown on chapter 3, to motivate and guide a database transition on the application proposed on the previous sections of this chapter.

As stated previously on 4.1.4, we want to check if the MySQL Database from Facebook Component needs to be transitioned, given the users are experiencing a degraded QoS in some parts of the application.

4.2.1 List application operations that are performed on database-level

According to the guidelines, the first step in a scenario where a database transition is being considered is to "List application operations that are performed on database level".

As the focus of our study is the Facebook component, it makes no sense to consider operations from other components, as "CRUD user account" and "CRUD terms to monitor".

This way, from the requirements presented on section 4.1.1, the operations that are related to the Facebook Component are **"Retrieve posts by ids"**, **"Classify posts (add tags)"** and **"Filter captured posts by filters"**. These are the operations that could possibly demand a transitioning process on the data layer of the component.

4.2.2 Define user-centered SLAs

The section 3.3.2 reveals that for each operation listed on the previous step, a set of SLAs should be proposed. This way, from interviews and surveys with application stakeholders, this network of SLAs could be proposed to the three operations that are subject to our study:

- **Retrieve posts by ids**:

  - Ideal threshold: 3 seconds
  - Tolerable threshold: 10 seconds
  - SLA Delta: 10x

- **Classify posts (add/remove tags)**:
  - Ideal threshold: 1.5 second
  - Tolerable threshold: 3 second
  - SLA Delta: 3x

- **Filter captured posts by filters**:
  - Ideal threshold: 3 seconds
  - Tolerable threshold: 10 seconds
  - SLA Delta: 3.3x

4.2.3 Define database-centered SLAs

As shown on previous sections, users and other application stakeholders are capable of identifying SLAs to the use cases of the application. On the other hand, the app engineers are responsible for defining SLAs at a database level.

These SLAs should be more "tight" than the SLAs defined by the users, as several other actions are also necessary to accomplish a use-case. In the context of a web application, for example, authentication tokens, filters, business logic and network delays are some of the factors that affect the execution time of a use case.

This way, a set of Database-oriented SLAs could be defined to the three proposed use-cases:

- **Retrieve posts by id**:
  - Ideal threshold: 1.0 seconds
  - Tolerable threshold: 4 seconds
  - SLA Delta: 4x
  - ROFR: 30%

- **Classify posts (add/remove tags)**:

  - Ideal threshold: 0.5 second
  - Tolerable threshold: 2 second
  - SLA Delta: 4x
  - ROFR: 30%

- **Filter captured posts by filters**:
  - Ideal threshold: 2 seconds
  - Tolerable threshold: 6 seconds
  - SLA Delta: 3x
  - ROFR: 15%

Thus, from the "Filter posts captured by filters" use case, it can be said that a request to filter posts at database level should take up to two seconds to assure that users won't be degraded performance caused by Database-related problems. If the query takes longer than six seconds to execute, users can get really frustrated about the slowness of the tool and this may have a big negative impact on the Quality of Service.

This SLA also reveals that there should be no problems if up to 15 % of the requests are executed between two and six seconds. This outlier data points may can be caused by expected instabilities on cloud providers.

4.2.4 Build database-level SLA log alerts

Parts of the application proposed on this chapter were implemented to assess the use of the guidelines. In a real-world scenario, log alerts could be completely developed within the source code of the application. Other services, such as New Relic, might also be used for this purpose.

It's also worth mentioning that the SLA checkers are only useful if database queries and processes are being executed by the users. In a simulated environment, user requests can be simulated by load test tools, as JMeter, or can be coded from scratch.

In our case study we built scenarios that simulate, at the same time, the requests that are sent by users and part of the part of the source code that analyzes SLA conditions and sends alerts if any SLAs is broken.

Through the implemented code it is possible to identify if is there any scenarios where the SLAs proposed on the previous chapter are not being fulfilled and consequently if is there any degraded QoS causd by database-related problems.

Building the test scenarios

Three scenarios - one for each operation of the Facebook component - were built to assess wether the SLAs for each operation are being met or not.

Through the following subsections we detail how each script was implemented and discuss the motivation to migrate from the relational architecture to a NoSQL strategy on the proposed application.

4.2.4.1 Retrieve posts by id

The first operation to be checked is to retrieve a set of posts, given their ids. This operation could called from a screen that presents a list of posts to the users, where they are able to check for details on each post, for example.

At database level, this operation can be seen as a simple SQL query, as revealed by listing 4.4.

Listing 4.4: SQL Query - Retrieve posts by ids

```
1 SELECT * FROM posts WHERE id IN (1,2,41,13,12903, ... ,435,31)
```

On the previous step, we have defined that a database-level SLA for this operation is composed by:

- Ideal threshold: 1.0 seconds
- Tolerable threshold: 4 seconds
- SLA Delta: 4x
- ROFR: 30%

To verify if the QoS of this operation is below expected, a test-routine was built according to the following steps:

- Generate a random list of 100 ids between between the range of posts that are stored on the database.

- Start a thread that opens opnnection to the database and retrieves the ResultSet with the list of posts.
- Wait for a random time between 30 to 300 milliseconds, to reproduce real-world scenario and avoid query flood on the database at once and start another thread of the same type until a total number of 100 threads are initiated.
- Repeat the steps above for 10 times.

Figures 13 and 14 present the Java implementation of the given algorithm. [1]

```
113 public class Ex01 {
114     public static void main(String[] args) {
115         int numQueriesPerExperiment = 100;
116         int numOfPostsToRetrieve = 100;
117         int numExperiments = 10;
118 
119         for (int i = 0; i < numExperiments; i++) {
120             for (int j = 0; j < numQueriesPerExperiment; j++) {
121                 RunnableDemo R1 = new RunnableDemo("Thread", numOfPostsToRetrieve);
122                 R1.start();
123                 try {
124                     int randomBetween30and300 = Util.generateRandomNumbers(30, 300, 1)[0];
125                     Thread.sleep(randomBetween30and300);
126                 } catch (InterruptedException e) {
127                     // TODO Auto-generated catch block
128                     e.printStackTrace();
129                 }
130             }
131         }
132     }
133 }
```

Figura 13: Experiment Setup

```
public void run() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://54.186.38.180:3306/mestrado", "root",
                "mestrado");

        Statement st = con.createStatement();
        String idList = Arrays
                .toString(Util.generateRandomNumbers(1, 3000000, numberOfIds))
                .replace("[", "(").replace("]", ")");
        String query = ("select * from post_all_pages_portugal_globo_pt where id in " + idList);

        long startTime = System.currentTimeMillis();
        ResultSet result = st.executeQuery(query);
        long estimatedTime = System.currentTimeMillis() - startTime;
        System.out.println((estimatedTime));
        addExecutionToArray(estimatedTime);
        con.close();

    } catch (ClassNotFoundException ex) {
        System.out.println("Driver not found");
    } catch (SQLException ex2) {
        ex2.printStackTrace();
    }
}
```

Figura 14: Main Thread - Run Method

Figure 15 presents the SLA checkers. As revealed on Chapter 3, alerts are triggered when any query exceeds the tolerable threshold or when the ROFR is over the desired

[1] This algorithm was run from the database server that host the database to assure that the measured time woud not be affected by network or other hardware-related delays.

level. We have also defined that a minimum number of 10 operations should be executed before any alert is sent, to avoid unnecessary alerts from unusual situations at the first time that a query is run.

```
34 class RunnableDemo implements Runnable {
35     private Thread t;
36     private String threadName;
37     private int numberOfIds;
38
39     static ArrayList<Long> executionArray = new ArrayList<Long>();
40
41     static int idealThreshold = 1000; // 1 sec
42     static int tolerableThreshold = 4000; // 4 sec
43     static float max_rofr = 0.30f; // 30%
44
45     static void addExecutionToArray(long executionTime) {
46         synchronized (executionArray) {
47             executionArray.add(executionTime);
48         }
49         checkSLAisBroken();
50     }
51
52     static void checkSLAisBroken() {
53         ArrayList<Long> executionsAboveIdealTheshold = new ArrayList<Long>();
54         synchronized (executionsAboveIdealTheshold) {
55             for (Long executionTime : executionArray) {
56                 if (executionTime > idealThreshold) {
57                     executionsAboveIdealTheshold.add(executionTime);
58                 }
59                 if (executionTime > tolerableThreshold) {
60                     System.err.println("Execution time over tolerable Threshold - ALERT");
61                 }
62             }
63             if ((executionsAboveIdealTheshold.size() / executionArray.size()) > max_rofr
64                     && executionArray.size() > 10) {
65                 System.err.println("ROFR Exceeded - ALERT");
66             }
67         }
68     }
```

Figura 15: Main Thread - Run Method

Figure 16 shows the execution time in milliseconds from the first scenario. On the chart, each line represents an experiment, where 100 queries of the same type as in listing 4.4 are sent to the MySQL database and the response time is registered.

As the ideal threshold to this operation states that all operations must be executed in up to 1000 milliseconds (a second), we can assure that the data layer is not a bottleneck to the users on this kind of operation. It is worth noticing that over 1.000 queries were executed on this scenario, and not a single query was over 1/10 of the desired ideal threshold.

Figure 17 shows the Java implementation of the java thread that makes the requests to the MySQL database and processes them on the SLA checkers.

On the Appendix A it is possible to visualize the raw data used on this first scenario.

4.2.4.2 Classify posts (add/remove tags) - Not Ready

> Inicialmente nao funciona, mas mudo o schema das tags e passa a funcionar

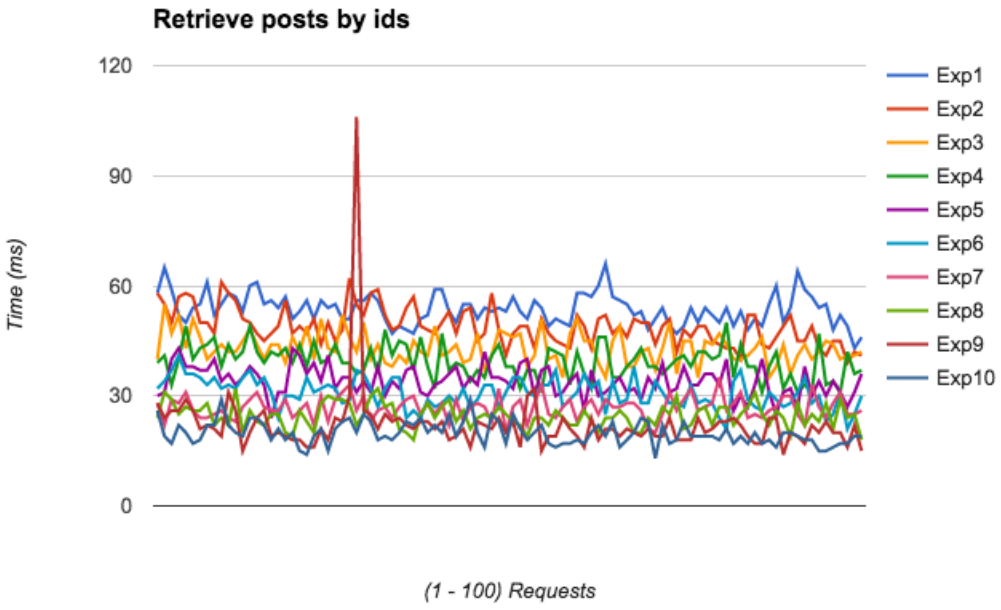

Figura 16: First Scenario - Retrieve post by ids.

```
static int idealThreshold = 1000; // 1 sec
static int tolerableThreshold = 4000; // 4 sec
static float max_rofr = 0.30f; // 30%

static void addExecutionToArray(long executionTime) {
    synchronized (executionArray) {
        executionArray.add(executionTime);
    }

    checkSLAisBroken();
}

static void checkSLAisBroken() {
    ArrayList<Long> executionsAboveIdealTheshold = new ArrayList<Long>();
    synchronized (executionsAboveIdealTheshold) {
        for (Long executionTime : executionArray) {
            if (executionTime > idealThreshold) {
                executionsAboveIdealTheshold.add(executionTime);
            }
            if (executionTime > tolerableThreshold) {
                System.out.println("Execution time over tolerable Threshold");
                System.exit(1);

            }
        }
        if ((executionsAboveIdealTheshold.size() / executionArray.size()) > max_rofr
                && executionArray.size() > 10) {
            System.out.println("ROFR Exceeded");
            System.exit(1);
        }
    }


}
```

Figura 17: Check SLA Violation.

4.2.4.3 Filter captured posts by filters - Not Ready

> Tem que transicionar para o Elasticsearch.

To verify this scenario within our database, we have implemented a runnable SLA checker in Python programming language. The algorithm, shown on Figure 18 performs fulltext-search operations on our MySQL database and verifies if the SLA is broken in this scenario.

First, we define several query sizes, and for each query size we generate random strings. Then, a query with of the following style is performed on our MySQL Database:

"SELECT COUNT(*) FROM posts WHERE message like %myQueryString%"

For each query size, we repeat the operation with a distinct word for three times (NUMBER_OF_ATTEMPTS_FOR_EACH_QUERY_LENGTH = 3). On our experiments, we found out that there's no noticeable difference between querying random generated ascii-chars and words from the dictionary.

```
1  mysql_queries = []
2  for query_length in [50, 30, 20, 10, 5, 2]:
3      for attempt in range(NUMBER_OF_ATTEMPTS_FOR_EACH_QUERY_LENGTH):
4          query_string = get_random_string_with_size(query_length)
5          print "..."
6          mysql_queries.append((query_length,search_time_mysql(query_string)))
7
8  print "MySQL Queries: %s" % mysql_queries
9  print "MySQL SLA is broken? %s" % slaIsBroken(mysql_queries,MAX_SECONDS)
```

Figura 18: Runnable SLA v0.1 .

We have tested the scenario with [100, 1000,10000, 100000, 100000, 1000000, 2000000 and 30000] posts within our dataset. Execution reports shows us that, indeed, the first SLO is broken with a data-size of 1000000 (one million) posts.

Execution reports for each of these dataset values is available on Appendix B.

Dentre as opcoes para melhorar o processo de search, ele aponta duas solucoes: (i) "To support full-text search, we needed to use the MySQL MyISAM storage engine. This has major downsides, the primary one being full table locks: when a table is updated, no other changes to that table can be performed." A outra saida era (ii) "We ended up doing this. It was a fairly simple step and allowed us to switch to the InnoDB engine on the master, eliminating the table lock issues.

This bought us some time, but it wasn't a long-term solution: we basically were rolling our own search and this frequently involved complex queries that third-party search libraries could perform more efficiently. We ended up with massive queries composed of many JOINs plus AND/ORs - these aren't easy to maintain.

Besides query complexity, it's tough to beat the performance of a dedicated search

solution. Our tables have considerable update activity, so this would result in sometimes-significant performance issues. ” Outros contra-pontos para implementar fulltext search em MySQL apontado por XXX, XXX and XXX sao X Y e Z. Falta terminar essa parte.

4.2.5 Verify SLA violation - Not ready

Para resolver o problema apontado no passo anterior, movemos a estrutura das tags para uma tabela separada.

On Figure 19 it is possible to verify the elements that compose a **Tag** record. Each tag has a unique ID - *idtags*, a *post_id* (foreign key to the post table), a *tag_user* (the username of the user who has tagged this post) and *tag_name*, the subject of this post.

The Tags table is also denormalized, as *tag_user* and *tag_name* are entities that should be represented on separate tables on a normalized schema.

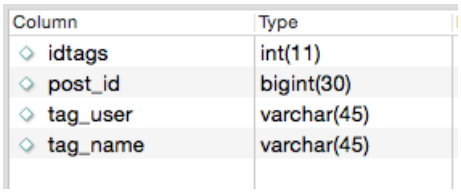

Figura 19: Tags table.

Tags information is purposely redundant. This way, it is possible to retrieve the tags of a post from a **JOIN** query between the Tags and Post tables and by searching it within the *tags* column of the Posts table. This data-redundancy was planned to test which option has better performance on a production scenario.

4.2.6 Propose architectural changes at database-level - Not ready

Apache Solr, Lucene, Amazon Cloudsearch and Elasticsearch are **Search Engines** that provide fulltext-search as master-features.

Elasticsearch (a.k.a. Elastic) was seemed to be a good alternative to the problem that we were facing with MySQL. Some discussion forums were consulted and comparisons and studies were considered on our research (STACKOVERFLOWELASTIC, 2015) (SOLRVSELASTICSEARCH, 2015) (QUORAELASTIC, 2015). With that, **step number four** is done.

A new data model should be proposed once the new database technology is chosen. As Elasticsearch stores data as JSON documents, the same structure of post that we had

on the posts table was transformed into a valid JSON document.

A new server with the same configuration of the one presented on the beginning of this chapter was provisioned and Elasticsearch[1.7.2] version was installed.

To dump the data from MySQL and import to Elasticsearch, a Python script was made (LEAL, 2015a). However, loading data was taking too long as the script didn't paralellize the bulk insert queries on Elasticsearch and database connection kept dropping. To overcome this situation,an open-source project that connects to MySQL via JDBC and imports data into Elasticsearch (JPRANTE, 2015) was used. With that, **the sixth step** is successfully executed.

A new runnable SLA was necessary to compare the execution time of the previous database architecture (MySQL) and the one that was proposed. We have joined both runnable SLAs into a single script, available on (LEAL, 2015b), **finishing the seventh step**.

It is possible to compare the Execution Report of both database architectures on Execution Report 02 (Appendix A).

With that, it is possible to show that the runnable SLA on the new architecture with all posts results in a significant performance improvement, proving that a database transition may be suitable to this scenario, **finishing Step 08**.

4.2.7 Map current schema & data on the proposed DB architecture

4.2.8 Process all DB operations from a historical point

5 Conclusions

Dizer que dava pra ao inves de tempo os slas serem orientados a acuracia tambem. Dizer de um caso: imagine uma aplicacao que diz quantos pontos tem dentro de um poligono. Com o banco tal a acuracia eh de X. Com o banco Y a acuracia eh de Z.

TODO: Fix References: https://www.evernote.com/l/AD9tKczyJiBK9KJkJ-s1eNh-AxhMlqz3SXA

The boundaries of this work could then be broadened to support the conception of a SLA-Guided process to support the migration/replacement of sotware components based on the cloud in future works.

Referências

ALBINSSON BARKAS, K. M. S. V. W. Blog, *Switching user database on a running system*. 2015. `https://labs.spotify.com/2015/06/23/user-database-switch/`.

ALLIANCE, C. S. *Security Guidance for Critical Areas of Focus in Cloud Computing*. [S.l.]: Cloud Security Alliance, 2009.

AMAZONRDS. *SLA for Cloud Services*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://aws.amazon.com/pt/rds/sla/>.

ANDRIEUX, A. et al. *Web Services Agreement Specification (WS-Agreement)*. [S.l.], set. 2005. Disponível em: <http://www.ggf.org/Public_Comment_Docs/Documents/Oct-2005/WS-AgreementSpecificationDraft050920.pdf>.

APPENGINE, G. *Google App Engine*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://cloud.google.com/appengine/docs>.

APPSEE. 2014. `https://www.appsee.com/`.

ARMBRUST, M. et al. *M.: Above the clouds: A Berkeley view of cloud computing*. [S.l.], 2009.

AXELOS. *Best Management Practice portfolio: common glossary of terms and definitions*. out. 2012. Disponível em: <http://www.axelos.com/gempdf/Axelos_Common_Glossary_2013.pdf>.

AZURE, M. *Azure*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://azure.microsoft.com>.

BAHL, S. Mysql vs neo4j vs mongodb an application centric comparison. 2014.

BEANSTALK, A. E. *Amazon Elastic Beanstalk*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://aws.amazon.com/pt/documentation/elastic-beanstalk/>.

BLANK, S. *The four steps to the epiphany*. [S.l.]: K&S Ranch, 2013.

BRIN, S. *Breaking Page's Law*. 2009. Disponível em: <https://www.youtube.com/watch?v=4kty5YNOaaw>.

BUILTWITH.COM. *CMS Ranking - 2015*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://trends.builtwith.com/cms>.

BUZZMONITOR. *Buzzmonitor*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://getbuzzmonitor.com>.

CHELIMSKY, D. et al. *The RSpec book: Behaviour driven development with Rspec, Cucumber, and friends*. [S.l.]: Pragmatic Bookshelf, 2010.

CLARK, J. Blog, *Google swaps out MySQL, moves to MariaDB*. 2013. `http://www.theregister.co.uk/2013/09/12/google_mariadb_mysql_migration/`.

COMPOSE.IO. *SLA for Cloud Services*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://docs.compose.io/enhanced/sla.html>.

CONNOLY D. FOX, A. e. a. *A View of Cloud Computing*. April 2010. [Online; posted 01-April-2010]. Disponível em: <"http://delivery.acm.org/10.1145/1730000/1721672/p50-armbrust.pdf">.

COUNCIL, T. P. P. Tpc-h benchmark specification. *Published at http://www. tcp. org/hspec. html*, 2008.

COURSERATECH. *Coursera's Adoption of Cassandra*. 2014. [Online; accessed 22-November-2015]. Disponível em: <https://tech.coursera.org/blog/2014/09/23/courseras-adoption-of-cassandra/>.

DATADOG. 2014. `https://www.datadoghq.com/`.

DHANDALA, N. *3 signs that you might need to move to NoSQL*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://cbblog.azurewebsites.net/3-signs-that-you-might-need-to-move-to-nosql/>.

DUAN, Y. et al. Everything as a service (xaas) on the cloud: Origins, current and future trends. In: *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. [S.l.: s.n.], 2015. p. 621–628.

EC2, A. *Amazon EC2*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://aws.amazon.com/en/ec2/>.

ELASTIC. *Elastic*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://elastic.co>.

ELLINGWOOD, J. *How To Set Up a Remote Database to Optimize Site Performance with MySQL*. 2014. [Online; accessed 22-November-2015]. Disponível em: <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-remote-database-to-optimize-site-performance-with-mysql>.

ENDPOINT. *Benchmarking Top NoSQL Databases*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf>.

FLOCKDB, T. *Twitter FlockDB*. 2014. [Online; accessed 22-November-2015]. Disponível em: <https://github.com/twitter/flockdb>.

FOWLER, M.; HIGHSMITH, J. The agile manifesto. *Software Development*, [San Francisco, CA: Miller Freeman, Inc., 1993-, v. 9, n. 8, p. 28–35, 2001.

GADE, K. Blog, *Twitter Search is Now 3x Faster*. 2011. `https://blog.twitter.com/2011/twitter-search-now-3x-faster`.

GLASS, R. Frequently forgotten fundamental facts about software engineering. *Software, IEEE*, v. 18, n. 3, p. 112–111, May 2001. ISSN 0740-7459.

HAN, J. et al. Survey on nosql database. In: *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*. [S.l.: s.n.], 2011. p. 363–366.

HAO, H. *Learning Python - Databases Chapter*. 2014. [Online; accessed 22-November-2015]. Disponível em: <http://learning-python.readthedocs.org/en/latest/chapter16/README.html>.

HAO, H. *Quora question: How do I migrate data from a MongoDB to MySQL database? Can it be done in a real-time scenario? What are the pros and cons for each migration? Which one do you advice? What is your experience? Any reference DB expert who can do it?* 2014. [Online; accessed 22-November-2015]. Disponível em: <https://www.quora.com/How-do-I-migrate-data-from-a-MongoDB-to-MySQL-database-Can-it-be-done-in-a-real-time-scenario-What-are-the-pros-and-cons-for-each-migration-Which-one-do-you-advice-What-is-your-experience-Any-reference-DB-expert-who-can-do-it>.

HAO, H. *Quora question: Why has Coursera migrated from MongoDB to MySQL?* 2014. [Online; accessed 22-November-2015]. Disponível em: <https://www.quora.com/Why-has-Coursera-migrated-from-MongoDB-to-MySQL>.

INSTACLUSTR. *Instaclustr Inc*. 2014. [Online; accessed 22-November-2015]. Disponível em: <https://www.instaclustr.com/>.

JPRANTE. *Elastic JDBC*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://github.com/jprante/elasticsearch-jdbc>.

KEPES, B. Understanding the cloud computing stack. *Rackspace White Paper*, 2011.

KLEMS, M.; BERMBACH, D.; WEINERT, R. A runtime quality measurement framework for cloud database service systems. In: *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*. [S.l.: s.n.], 2012. p. 38–46.

KOUKI, Y.; LEDOUX, T. CSLA : a Language for improving Cloud SLA Management. In: *International Conference on Cloud Computing and Services Science, CLOSER 2012*. Porto, Portugal: [s.n.], 2012. p. 586–591. Disponível em: <https://hal.archives-ouvertes.fr/hal-00675077>.

KOUKI, Y. et al. A language support for cloud elasticity management. In: *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. [S.l.: s.n.], 2014. p. 206–215.

LEAL, F. *Python to Elasticsearch importer*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://github.com/fabiosl/ufrn-masters-research/blob/master/exp01/python-mysql-to-es-importer/importer.py>.

LEAL, F. *Runnable SLA 01*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://github.com/fabiosl/ufrn-masters-research/blob/master/experiments/exp01_main.py>.

LEAL, F.; MUSICANTE, M. Todo: Update this reference - using sla to guide database transition to nosql on the cloud: a systematic mapping study. In: *AICCSA 2015*. ?: ?, 2015. (?), p. ? ISBN ? Disponível em: <?>.

LEAVITT, N. Will nosql databases live up to their promise? *Computer*, v. 43, n. 2, p. 12–14, Feb 2010. ISSN 0018-9162.

LEE, Y. C. et al. Profit-driven service request scheduling in clouds. In: *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2010. (CCGRID '10), p. 15–24. ISBN 978-0-7695-4039-9. Disponível em: <http://dx.doi.org/10.1109/CCGRID.2010.83>.

LOCAWEBSLA. *SLA for Cloud services*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://wiki.locaweb.com.br/pt-br/SLA_nos_planos_compartilhados>.

LOMBARDO, S.; NITTO, E. D.; ARDAGNA, D. Issues in handling complex data structures with nosql databases. In: IEEE. *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*. [S.l.], 2012. p. 443–448.

MASSOL, V.; HUSTED, T. *Junit in action*. [S.l.]: Manning Publications Co., 2003.

MENTION. *Mention*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://mention.com>.

MOODLE. *Moodle - Course Management System*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://moodle.org/>.

MOORE, G. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, v. 86, n. 1, p. 82–85, Jan 1998. ISSN 0018-9219.

MOUSSA, R. Benchmarking data warehouse systems in the cloud. In: *Computer Systems and Applications (AICCSA), 2013 ACS International Conference on*. [S.l.: s.n.], 2013. p. 1–8. ISSN 2161-5322.

MYSQLRECOVERY. *7.5 Point-in-Time (Incremental) Recovery Using the Binary Log*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://dev.mysql.com/doc/refman/5.7/en/point-in-time-recovery.html>.

NEPAL, S.; ZIC, J.; CHEN, S. Wsla+: Web service level agreement language for collaborations. In: *Services Computing, 2008. SCC '08. IEEE International Conference on*. [S.l.: s.n.], 2008. v. 2, p. 485–488.

NEWRELIC. 2014. `http://newrelic.com`.

NOSQL-ORG. *NoSQL-ORG*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://nosql-database.org>.

ORACLERECOVERY. *Performing Database Point-In-Time Recovery*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://docs.oracle.com/cd/B19306_01/backup.102/b14192/flashptr006.htm>.

PARIS, H. *Fantastic Elasticsearch plugin for Wordpress*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://wordpress.org/plugins/fantastic-elasticsearch/>.

PARISEAU, B.; JONES, T. *Cloud outage audit 2014 reveals AWS on top, Azure down*. 2014. [Online; accessed 22-November-2015]. Disponível em: <http://searchcloudcomputing.techtarget.com/news/2240237323/Cloud-outage-audit-2014-reveals-AWS-on-top-Azure-down>.

PDFTOKENSMATCHER. 2015. https://github.com/fabiosl/pdf-tokens-finder. Accessed: 2015-05-01.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. Swinton, UK, UK: British Computer Society, 2008. (EASE'08), p. 68–77. Disponível em: <http://dl.acm.org/citation.cfm?id=2227115.2227123>.

POSTGRESRECOVERY. *Continuous Archiving and Point-in-Time Recovery (PITR)*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://www.postgresql.org/docs/9.1/static/continuous-archiving.html>.

POWELL, T. C.; DENT-MICALLEF, A. Information technology as competitive advantage: The role of human, business, and technology resources. *Strategic management journal*, v. 18, n. 5, p. 375–405, 1997.

QI, M. et al. Big data management in digital forensics. In: *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*. [S.l.: s.n.], 2014. p. 238–243.

QUINN, J.; DOORLEY, T.; PAQUETTE, P. Technology in services: rethinking strategic focus. *Sloan Management Review, Win-ter1990QuinnWinterSloan Management Review1990*, 2013.

QUORAELASTIC. *Quora - Differences between Elastic, Solr and SolrCloud*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://www.quora.com/What-are-the-main-differences-between-ElasticSearch-Apache-Solr-and-SolrCloud>.

RACKSPACE. *Rackspace*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://www.rackspace.com/>.

RANA, O. et al. Managing violations in service level agreements. In: *Grid Middleware and Services*. Springer US, 2008. p. 349–358. ISBN 978-0-387-78445-8. Disponível em: <http://dx.doi.org/10.1007/978-0-387-78446-5_23>.

RANKING, D. *DB Engines Ranking*. maio 2014. Disponível em: <http://db-engines.com/en/ranking>.

RANKINGCHART. *DB Engines RankingChart*. 2015. Disponível em: <http://db-engines.com/en/ranking_categories>.

RAPP, B. Blog, *Goodbye, John McCarthy*. 2011. `http://cavewall.jaguardesignstudio.com/2011/10/24/goodbye-john-mccarthy/`.

REDMINE. *Redmine Project Management Software*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://www.redmine.org/>.

SADALAGE, P. *NoSQL Databases: An Overview*. 2014. [Online; accessed 22-November-2015]. Disponível em: <https://www.thoughtworks.com/insights/blog/nosql-databases-overview>.

SADALAGE, P. J.; FOWLER, M. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. [S.l.]: Pearson Education, 2012.

SAKR, S.; LIU, A. Sla-based and consumer-centric dynamic provisioning for cloud databases. In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. [S.l.: s.n.], 2012. p. 360–367. ISSN 2159-6182.

SANDERS, G.; SHIN, S. Denormalization effects on performance of rdbms. In: *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. [S.l.: s.n.], 2001. p. 9 pp.–.

SHI, W. et al. An online auction framework for dynamic resource provisioning in cloud computing. *Networking, IEEE/ACM Transactions on*, PP, n. 99, p. 1–1, 2015. ISSN 1063-6692.

SHINDER, D. Security in the cloud: Trustworthy enough for your business? *http://bit.ly/1jhbnkC*, v. 11, 2010. Disponível em: <http://bit.ly/1jhbnkC>.

SHVACHKO, K. et al. The hadoop distributed file system. In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. [S.l.: s.n.], 2010. p. 1–10.

SOLAR, G. V. *Addressing data management on the cloud: tackling the big data challenges*. maio 2014. Disponível em: <http://ceur-ws.org/Vol-1087/keynote4.pdf>.

SOLRVSELASTICSEARCH. *Solr. vs Elasticsearch*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://solr-vs-elasticsearch.com/>.

SPECTOR, A.; NORVIG, P.; PETROV, S. Google's hybrid approach to research. *Commun. ACM*, ACM, New York, NY, USA, v. 55, n. 7, p. 34–37, jul. 2012. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/2209249.2209262>.

SPROUTSOCIAL. *SproutSocial*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://sproutsocial.com>.

STACKOVERFLOWELASTIC. *Stack Overflow - Solr. vs Elasticsearch*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://stackoverflow.com/questions/10213009/solr-vs-elasticsearch>.

STANOEVSKA-SLABEVA, K.; WOZNIAK, T. *Grid Basics*. [S.l.]: In: Stanoevska-Slabeva, K., Wozniak, T., and Ristol, S., Grid and Cloud Computing A Business Perspective on Technology and Applications, 2009.

STURM, R.; MORRIS, W.; JANDER, M. {Foundations of Service Level Management}. Sams publishing, 2000.

WIKIPEDIA. *John McCarthy (computer scientist)*. 2015. [Online; accessed 22-November-2015]. Disponível em: <https://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist)>.

WIKIPEDIA-COURSERA. *Coursera - Wikipedia*. 2014. [Online; accessed 22-November-2015]. Disponível em: <https://en.wikipedia.org/wiki/Coursera>.

WIRTH, N. A plea for lean software. *Computer*, v. 28, n. 2, p. 64–68, fev. 1995.

WORDPRESS. *Wordpress CMS*. 2015. [Online; accessed 22-November-2015]. Disponível em: <http://wordpress.com>.

WU, L.; BUYYA, R. Service level agreement (sla) in utility computing systems. *IGI Global*, 2012.

XIONG, P. et al. Activesla: A profit-oriented admission control framework for database-as-a-service providers. In: *Proceedings of the 2Nd ACM Symposium on Cloud Computing*. New York, NY, USA: ACM, 2011. (SOCC '11), p. 15:1–15:14. ISBN 978-1-4503-0976-9. Disponível em: <http://doi.acm.org/10.1145/2038916.2038931>.

ZHAO, L.; SAKR, S.; LIU, A. A framework for consumer-centric sla management of cloud-hosted databases. *Services Computing, IEEE Transactions on*, PP, n. 99, p. 1–1, 2013. ISSN 1939-1374.

ZHU, J.; WANG, A. Data modeling for big data. *CA, Beijing*, Citeseer, 2012. Disponível em: <"http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.434.7867&rep=rep1&type=pdf#page

APÊNDICE A – Systematic Mapping

Using SLA to guide database transition to NoSQL on the cloud: a systematic mapping study

Fabio Leal and Martin A. Musicante
Computer Science Department, Federal University of Rio Grande do Norte, Natal, Brazil
Email: sousaleal.fabio@gmail.com, mam@dimap.ufrn.br

***Abstract*—Cloud computing became a reality, and many companies are now moving their data-centers to the cloud. A concept that is often linked with cloud computing is Infrastructure as a Service (IaaS): the computational infrastructure of a company can now be seen as a monthly cost instead of a number of different factors. Recently, a large number of organizations started to replace their relational databases with hybrid solutions (NoSQL DBs, Search Engines, ORDBs). These changes are motivated by (*i*) performance improvements on the overall performance of the applications and (*ii*) inability to a RDBMS to provide the same performance of a hybrid solution given a fixed monthly infrastructure cost. However, not always the companies can exactly measure beforehand the future impact on the performance on their services by making this sort of technological changes (replace RDBMS by another solution). The goal of this systematic mapping study is to investigate the use of Service-Level-Agreements (SLAs) on database-transitioning scenarios and to verify how SLAs can be used in this processes.**

Keywords*—*Transition to Cloud, NoSQL, Systematic Mapping.

I. Introduction

The adoption of cloud solutions is growing fast among organizations [1]. Centralized (mostly mainframe) technology is being replaced by distributed and more flexible forms of data storage and processing. This change of paradigm is motivated by the necessity to improve the use of resources, as well as by the increasing velocity in which data is produced.

In this scenario, transitions must take into account the quality of the service delivered by the new solutions. The notion of Service-Level-Agreement (SLA) [2] may be used as a parameter in this context. SLAs are widely used to to provide service thresholds between clients and providers and are present in almost every service contract over the internet.

SLAs or OLAs - Operational Level agreements, are particularly useful to guide the process of choosing the most convenient service from a pool of service providers.

In this paper, we survey the use of SLA on database-transitioning scenarios, trying to investigate how they might be used to help the execution of this process. Our study is performed using the systematic mapping [3] technique: A set of papers is retrieved from the most popular bibliography repositories; this set is then filtered according to predefined parameters and finally, the analysis of the remaining papers is guided by a small number of research questions.

This work was partly funded by CNPq (Brazil, PDE-201118/2014-9, PQ-305619/2012-8, INCT-INES-573964/2008-4), CAPES/CNRS (Brazil/France, SticAmSud 052/14) and CAPES/ANII (Brazil/Uruguay, Capes-UdelaR 021/10).

This paper is organized as follows: Section II presents some of the concepts that are related to the transition from a traditional setting to a cloud-aware one. Section III presents our research questions and each step of our survey. The outcomes of our Systematic Mapping study can be seen on Section IV.

II. The Technological Shift

On the early 90's it was commonplace for every Information Technology (IT) company to have its own Data Center with huge servers and mainframes. IT costs were high, and high-performance computing was available only for big companies, as data centers required a large physical infrastructure and have high costs for maintenance [4].

The regular way of building a web application was to use a client-server approach, where the server was a powerful (and expensive) machine. At the same time, new players, such as Google or Yahoo, were rising with bigger missions: *"to organize the world's information and make it universally accessible and useful"* [5]. The popularization of the internet use new ways of commerce exchange, yielding an explosion in the amount of data produced and exchanged. It was *just* impossible to store the petabytes of daily-generated data in a single server.

From this point on, the community realized the economical convenience of building and maintaining several low-performance servers, instead of a single high-performance one, even if this this requires a change of culture in the administration of the new datacentres. The new approach is also incompatible with the traditional way of building applications, that usually were designed to work on a single server and database.

Several research initiatives were conducted in this area and a common solution was rising: to distribute data storage and processing. Google, Yahoo and other big IT players helped to build open source tools to make this approach possible, like Hadoop [6].

This revolution brought to life the notion of *Cloud Computing*, together with new concepts, such as Infrastructure as a Service *(IAAS)*, Platform as a Service *(PAAS)* and Software as a Service *(SAAS)* [7]. According to [7], *Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services.*

Data Integration & Polyglot Persistence: On the last years, the number of Data Base (DB) Engines grew like never before [8]. Along with the NoSQL (Not only SQL) movement

and expansion of Social Networks, new concepts for Database Models appeared, like Document Store, Search Engines, Key-Value store, Wide Column Store, Multi-Model and Graph DBMS. In [8] a ranking of the most popular DB engines is presented.

Today, instead of having a single Relational Database Management System (DBMS) for the whole application, it is efficient and cost-effective to have several Data Base Engines, one for each type of data that the application handles. This concept is called *Polyglot Persistence* [9].

As [10] illustrates, polyglot persistence is very useful in the context of e-commerce applications that deal with a catalog, user access logs, financial information, shopping carts and purchase transactions, for example. The notion of polyglot persistence is built upon the observation that the *nature* of each data type is significantly different (i.e: user logs imply high volume of writes on multiple nodes, shopping carts need high availability and user sessions require rapid access for reads and writes).

As computing services started to decentralize, developers started to build applications that depended of several data-sources. By this time the use of Web Services and Service Oriented Architecture (SOA) became more popular [4].

Service Level Agreements (SLAs): According to *ITILv3's* official glossary [11], a Service Level Agreement (SLA) is "*an agreement between an IT service provider and a customer. A service level agreement describes the IT service, documents service level targets, and specifies the responsibilities of the IT service provider and the customer.*"

The agreement consists on a set of measurable constraints that a service provider must guarantee to its customers. In practical terms, it is a document that a service provider delivers to its consumers with minimum quality of service (QoS) metrics. If the service is delivered with a lower QoS than is promised on the SLA, consumers may be refunded or earn benefits that were accorded beforehand.

Systematic Mappings: According to [3], "*A software engineering systematic map is a defined method to build a classification scheme and structure a software engineering field of interest.*" Systematic Mapping studies provide a global view of a given research field and identify the quantity, results, and the kinds of researches in this field.

A Systematic map is composed by a number of steps (Figure 1).

On the first step, "Definition of Research question", the questions that must be answered on the survey are defined. On the "Review Scope" step, researchers target the papers/journal sources that will be taken into consideration on the systematic map. After that, the "Search" step is done using a set of predefined search engines and a body of papers ("All papers") is retrieved. After an initial "Screening of the papers", the "Relevant papers" are chosen according to inclusion and exclusion criteria defined by the research team. At this point, the papers that will participate of the study are selected. The selection is based on the title, abstracts and keywords of each paper ("Keywording using Abstracts"). After that, a "Classification Scheme" is built, defining different points-of-view (or facets) from which the body of papers will be classified. After matching each paper with the classification schema ("Data Extraction and Mapping Process"), the systematic mapping is performed. In this phase the relationships between the collected data (in the light of the classification scheme) are used to answer the research questions.

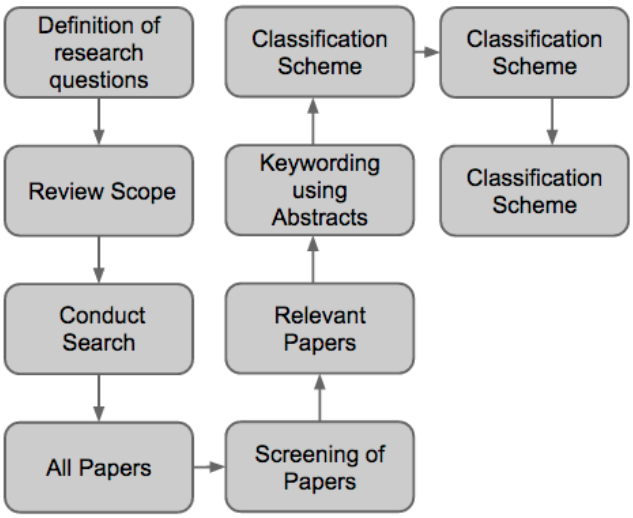

Fig. 1. Systematic Mapping Steps [3].

In the next section we build a systematic mapping to evaluate the use of SLA in database transition scenarios, from relational database management systems to NoSQL.

III. A Systematic Mapping

In this section we develop our systematic mapping, in the way described in section II.

We start by defining our research scope and by pointing out the questions to be answered by the end of the survey (Section III-A), followed by the selection of keywords that composed the search string (Section III-B). The other steps of the systematic mapping procedure follow.

After that, we defined a classification schema and classified the relevant papers. These steps (screening of papers to classification scheme) were not linear as shown on Figure 1. We had to iterate on the screened papers a few times to find the ideal classification schema to our systematic map.

A. Definition of the research questions

As we wanted to investigate the way in which SLAs has been used in database transitioning processes, we proposed three main research questions and two associated questions, as follows:

RQ1) What are the reasons to change from RDBMSs to NoSQL solutions? This question is about *why* technological changes are needed on Information Systems and what is the *motivation* behind them.

AQ1.1) What are the pros and cons to migrate from RDBMSs to NoSQL solutions? This question is particularly

important, as we also want to *point out the down sides* of DB migrations.

AQ1.2) How can we measure the overall improvements promised by this change? After a migration is performed, it is also important to *measure the benefits* of this migration, and this question evidences this point.

RQ2) How can SLAs be used to guide database transitioning processes from RDBMSs to NoSQL databases in cloud-based apps? This is one of the main questions of our study, as we want to evidence the state-of-art on the use of SLAs in migration scenarios.

RQ3) Is there a standard representation of SLAs in cloud services? We also try to search for standards on the representation of SLAs to verify if future works can be done under this field.

These are relevant questions for our survey as our focus is to determine how SLAs can be used in Database migrations scenarios. Our questions are intended to cover *why* it is good to migrate from RDBMSs to a NoSQL solution and *how* we can do it. In the end of our survey we also reveal what are the most popular technologies related to migrations.

B. Definition of keywords

Our research scope can be splitted in three main categories:

1) Databases: As we wanted to investigate RDBMS to NoSQL transitions, we defined that a representative query string for this category would be

NoSQL AND (rdbms OR relational)

2) Migration: A number of terms terms were oftenly linked with database migration and transition scenarios. Most of them, however, are consistently represented by the radicals "migr" and "transit". Thus the representative search string for this category is defined by:

migr OR transit**

3) Service level agreements: Service Level Agreements can also be represented by its acronym (SLA). A search string for this category is defined as

"Service Level Agreement" OR SLA

As we defined the three main categories of interest, what we wanted to survey is their conjunction. In this way, the final search string can be represented as:

(migr OR transit*) AND (NoSQL AND (rdbms OR relational)) AND (SLA OR "Service Level agreement")*

C. Conduct Search for Primary Studies (All Papers)

The next step of our systematic map was to identify relevant publishers. We surveyed academia experts and initially chose five main sources: **Springer, ACM, Sciencedirect, IEEE and Elsevier**. All of these publishers have relevant publications about databases and Service Level Agreements, as they index publications from reputable international conferences & journals.

Each of these publishers had their own search engine, however some inconsistent results were obtained when querying some of them directly. For instance, on one occasion IEEE search engine returned 0 results to the query string "*changes AND database AND nosql AND sla*". When we searched the same query on Google Scholar filtering *only IEEE publications* we found **90** results. We supposed that this erratic behavior is due to network traffic conditions. In order to mitigate the risks of having different results for the same query on different search engines, we used Google Scholar as a meta-search engine. It is important to mention that the use of Google Scholar made it possible to enlarge our search space. Our scope is not limited to the list of publishers identified above; these publishers represent only the minimal coverage of the literature in this systematic map.

During this step, we also noticed that 2009 would be the starting year of our survey, since this was the year when the term NoSQL was reintroduced to the community, by Johan Oskarsson (Last.fm) [12].

D. Search Strategy

Google Scholar returned 74 results for our search string. We have also manually included other particularly relevant publications that were picked from selected sources, such as [13], a master thesis about Relational Database Schema Migration to NoSQL.

These publications were either not published by the selected publishers or were not indexed by Google Scholar.

Our search strategy was composed by the following steps:

Step 1 - Basic Search: Search publications from 2009 to 2015 that matched the query string and add other relevant publications manually. This step was performed from 01 April 2015 to 11 April 2015 and returned 80 publications.

Step 2 - Dump the retrieved results on a spreadsheet. This spreadsheet is publicly available on [14].

Step 3 - Screening for keywords in of title and abstract: On the spreadsheet each publication has title, abstract, year and referenced URL. Title and abstract were considered on the initial screening. On this step we discarded publications that are notably not related to the topic of our study.

Step 4 - Apply inclusion/exclusion criteria: The inclusion/exclusion criteria are shown below. We first applied the inclusion criteria over the selected works, and then exclusion criteria removed the out of scope publications. Only papers clearly not relevant for the purposes of the study were removed. The publications that were considered on this study are marked on the spreadsheet with a green background.

1) Inclusion Criteria:
   - The publication is about a migration from RDBMS to a NoSQL technology;
   - The main focus of the publication is on RDBMs or NoSQL systems;
   - The publication makes use or references a SLA.

2) Exclusion Criteria
   - Non-English written publications;
   - Access-restriction to the original publication (could not access the source of the publication);
   - Non-technical publications;
   - The work is about migrations within the same database DBMS;
   - The work is not related to databases or SLAs.
   - The work is related to databases but no comparison is made between two technologies.

A total of 35 publications were selected after this phase.

Step 5 - Fast Reading of the papers: We've performed a fast read of all publications that were not excluded on the previous step. This helped us to classify each paper accurately. This step is extensively detailed on the next section.

E. Classification of the Papers

We classified each selected publication in three facets: **Contribution Type**, **Technologies used** and **SLA representation**.

1) **Contribution type**: On the initial screening of papers we have defined 5 main types of contribution to our study. Other systematic mapping studies, such as [15] and [16] use a similar classification schema for contribution type.
   - Benchmark
   - Migration Experience Report
   - Bibliographical Review
   - Tool
   - Framework / Process

   When a migration of databases is described in a publication, we also classify specific data to answer the questions of this study: Source and Target technology of the migration, motivation to migrate and "Uses SLA or other artifact to validate the migration?";
2) **Technologies used**: To classify and rank the technologies mentioned on each paper we have developed a *"Batch-PDF-Tokens-Matcher"*. This tool matches a list of strings against a list of PDF files. The tool was made open-source and is publicly available on [17]. In our case, we wanted to find the most cited technologies on relevant publications. As it was not possible to include all database types on this category, we chose to match only the 250 most-popular databases, ranked and made publicly available by [8].
3) **SLA representation**: One of the questions that our study wanted to answer was RQ3 (Is there a standard representation of SLAs in cloud services?). There are several ways to represent an SLA, and we clusterized some of these ways. It is also possible that no information is given about SLA representation on the publication, so we added two subcategories to this facet: "No SLA is used" and "Missing information".
   - Tool/Code/Database records
   - Language
   - Missing Information
   - No SLA is used

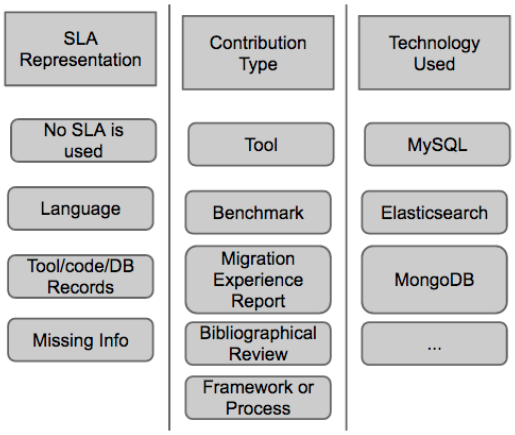

Fig. 2. Classification scheme for selected publications.

A graphical representation of our classification schema is detailed on Figure 2. We have listed only a subset of the several technology types that were found on the review.

IV. Outcomes

As mentioned in Section III-D, 74 publications were found matching our query string on IEEE, Elsevier, ACM, Springer and Sciencedirect using Google Scholar as a meta-search engine. We have manually added other 5 publications to our study. These publications were either not published by the selected publishers or were not indexed by Google Scholar.

35 out of the total of 79 publications were selected, after the screening of abstract and analysis of the inclusion/exclusion criteria[1].

The considered publications are: [18] [2] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [21] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [1] [49].

The results are given in two steps: *Frequency Analysis between relevant criteria* and *Answers to research questions*.

A. Frequency Analysis between relevant criteria

Each analysis is presented below and is followed by a brief interpretation of the result.

- Publications by year: The publications by year table - Figure 3 - shows us that 2013 was the year when most of the publications were made on this research area. As we have just reached the middle of 2015 it is expected that not many publications are indexed on the search engines on this year.
- Frequency - Publication type vs Years: An interesting pattern can be found on the table Publication Type vs Years (Figure 4); "Migrations Experience Reports" were the main type of publication that our study aimed to discover, but only 3 publications of this type were found on our systematic map.

[1] A column in our spreadsheet [14] contains the justification for the rejection of each discarded publication.

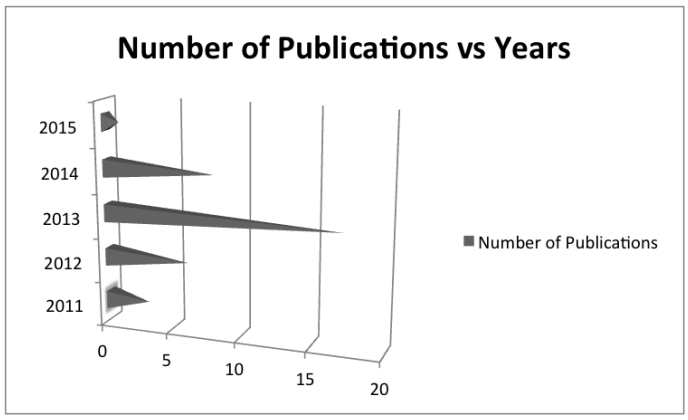

Fig. 3. Publications by year.

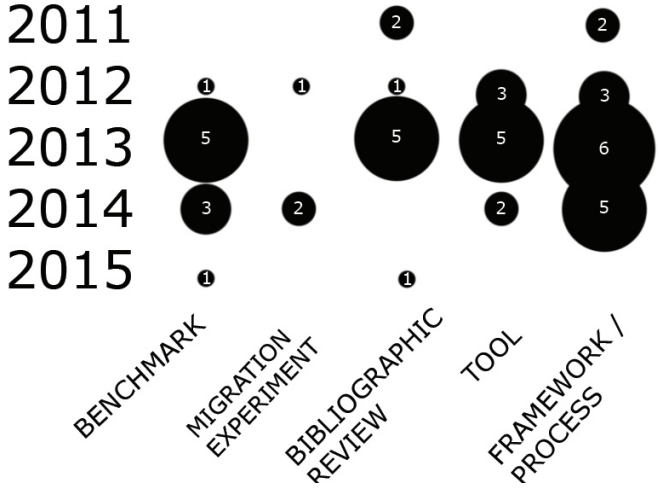

Fig. 4. Frequency - Publication type vs Years.

- Publications by country: Figure 5 shows that USA, Australia and Romania were responsible for almost 50% of the total selected publications. The other half of publications were distributed among other 16 countries, proving that the research area is not being developed by a single research group.

- Popular Technologies: Figure 6 shows that Cassandra, MongoDB and MySQL were the most-cited databases on the papers that we analyzed. This chart can be used as a starting point to propose new works in the area of database migration. The full count of each technology is available on [50].

- SLA Representation: 29% of the publications didn't make any use of SLAs. Another big part - 32% - didn't even mention how the SLA is represented internally. 3 publications propose or use DSLs (Domain Specific Languages) to represent SLAs. This evidences a clear absence of standards when defining/using SLAs. The full table is presented on Figure 7. [50].

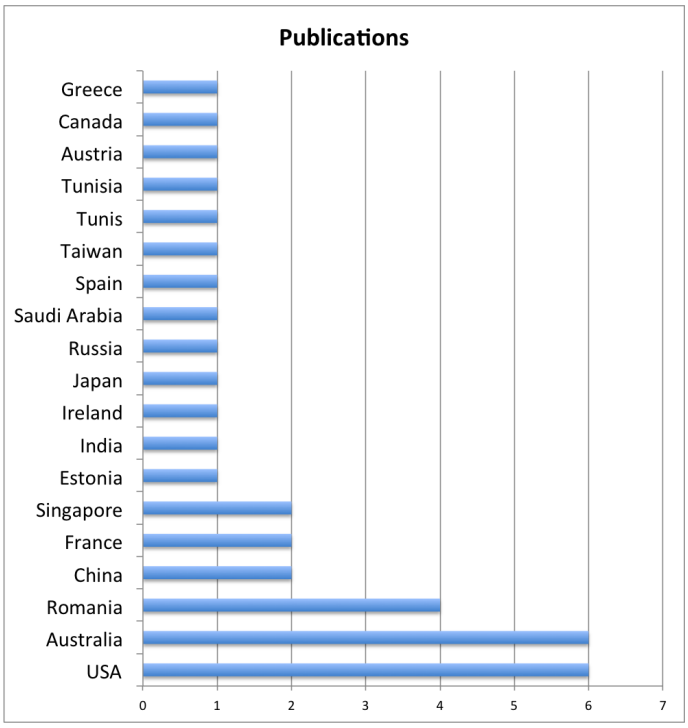

Fig. 5. Publications by country.

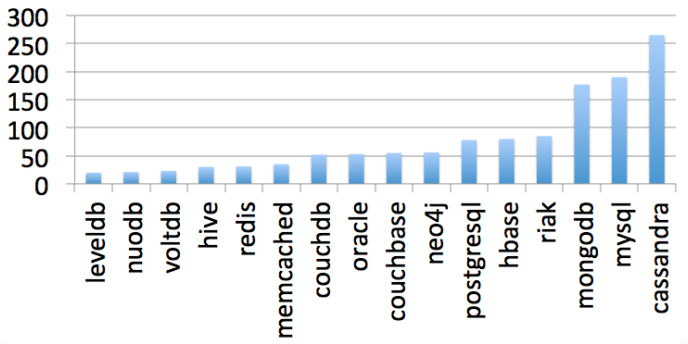

Fig. 6. Most-cited database technologies.

B. Answers to research questions

In this mapping study we have surveyed the state of the art in transitions from RDBMSs to NoSQL databases. We created a rigorous protocol which analyzed 79 publications to answer the research questions that we identified previously. We may consider the answer to these questions as the main outcome of this paper. The answers are summarized below:

RQ1-What are the reasons to change from RDBMSs to NoSQL solutions? and **AQ1.2**-What are the pros and cons to migrate from RDBMSs to NoSQL solutions? investigate the motivation of a database transition. We found a number of reasons to migrate from a relational database to a NoSQL/Hybrid model. As the migration is most of the time motivated by the benefits of these models, the answers to the questions RQ1 and AQ1.2 are presented together.

The publications that reference the benefits and disadvantages of NoSQL / Hybrid solutions were [46] [47] and [48].

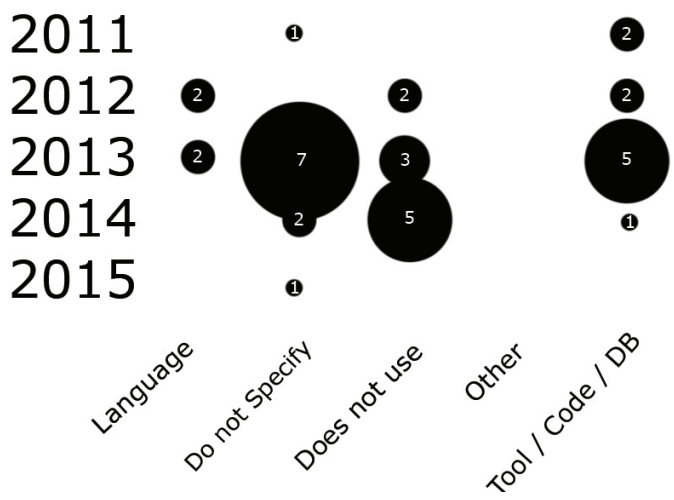

Fig. 7. SLAs representation.

Benefits::

- Segments of the data to be read and processed in parallel using a MapReduce framework, such as Hadoop;
- Schema-less data model;
- Support for large files;
- Scalability - relational databases tend to "Scale up", which is opposed to the "Scale horizontally" strategy used by hybrid solutions;

Disadvantages::

- A big disadvantage on moving from a RDBMS to a NoSQL/Hybrid solution is the need for changes on the data models of the application. Several publications address this problem, such as [46] [51] and [52]
- Another disadvantage of NoSQL and Hybrid solutions is the fact that these concepts are relatively new. As we mentioned previously, the term NoSQL was used to reference these types of databases for the first time in 2009 [12]. The relational model has been in use for more than 30 years;As it is a new concept, it is particularly hard to find developers with a large experience in NoSQL databases.

AQ1.2-How can we measure the overall improvements promised by this change?

No publication was found addressing the problem of measuring the overall improvements after a database transition. In fact, as it is shown on IV-A, there are few publications with the "Migration Experience Report" type.

Several benchmarking frameworks, such as TPC-H, TPC-DS and YCSB were identified [21] during our survey, though. These benchmarking frameworks could be a good starting point to develop new tools and specialized frameworks to solve this problem. This seems to be a promising research theme for future works.

RQ2-How can SLAs be used to guide database transitioning processes from RDBMSs to NoSQL databases in cloud-based apps?

A number of works were found relating SLAs with Quality of Service (QoS) and Quality of Experience (QoE). Several publications, such as [22], [53] and [54] propose a SLA-centric approach to monitor and control the performance of cloud-based apps. [2], [19], [55] and [36] propose SLA-centric/User-Centric solutions to monitor the performance of web applications. All these solutions are technology-agnostic and could be used to monitor the performance improvements promised by a database transitioning process.

The question RQ2 was subject of discussion with industry experts and it was found out that there are some services, such as New Relic [56], Appsee [57] and Datadog [58] that provide SLA-monitoring tools for web apps.

RQ3-Is there a standard representation of SLAs in cloud services?

The selected publications did not present a standardized and common representation for SLAs. In fact, 32% of the selected publications did not even mention how the SLA was represented. 29% represented the SLAs as tables/documents stored on databases without any sla-oriented guidelines. This evidences a clear absence of standards when defining/using SLAs.

[1] proposes SYBL: *An Extensible Language for Controlling Elasticity in Cloud Applications. SYBL allows specifying in detail elasticity monitoring, constraints, and strategies at different levels of cloud applications, including the whole application, application component, and within application component code.* In other words, SYBL can also be seen as a language to specify SLAs in cloud environments.

Specifically related to SLAs on Cloud Services, [49] and [59] propose CSLA (Cloud Service Level Agreement): a language for Cloud Elasticity Management. CSLA features concepts related to SLAs on Cloud service, as QoS/functionality degradation and penalty models. These features are very useful, as it allows providers to express contracts with fines and penalties.

Some other initiatives that tried to define the mechanisms by which Webservice SLAs are established are WS-Agreement [60] and WSLA [61]. Also, [62] presents a framework for managing the mappings of the Low-level resource Metrics to High-level SLAs (LoM2HiS framework).

It is worth to notice that future works on standardizing the representations of SLAs are needed.

V. Conclusions

By analyzing the charts and the answers to the research questions we can conclude that the research area of this mapping study is still not mature.

There is a lack of consensus and standards on this research area, as no official guidelines for migrating from a relational database to a NoSQL/Hybrid database were found. It was also not found a standard way to represent and create SLAs. These

two points seems to be promising research topics and will be covered on future works.

REFERENCES

[1] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "Sybl: An extensible language for controlling elasticity in cloud applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, May 2013, pp. 112–119.

[2] S. Sakr and A. Liu, "Sla-based and consumer-centric dynamic provisioning for cloud databases," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, June 2012, pp. 360–367.

[3] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE'08. Swinton, UK, UK: British Computer Society, 2008, pp. 68–77. [Online]. Available: http://dl.acm.org/citation.cfm?id=2227115.2227123

[4] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A berkeley view of cloud computing," Tech. Rep., 2009.

[5] A. Spector, P. Norvig, and S. Petrov, "Google's hybrid approach to research," *Commun. ACM*, vol. 55, no. 7, pp. 34–37, Jul. 2012. [Online]. Available: http://doi.acm.org/10.1145/2209249.2209262

[6] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, May 2010, pp. 1–10.

[7] A. e. a. CONNOLY, D. FOX, "A view of cloud computing - http://delivery.acm.org/10.1145/1730000/1721672/p50-armbrust.pdf," April 2010, [Online; posted 01-April-2010].

[8] "Db engines ranking," http://db-engines.com/en/ranking, Accessed: 2014-05-01.

[9] P. J. Sadalage and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.

[10] G. V. Solar, "Addressing data management on the cloud: tackling the big data challenges," May 2014. [Online]. Available: http://db-engines.com/en/ranking

[11] Axelos, "Best management practice portfolio: common glossary of terms and definitions," Oct. 2012. [Online]. Available: http://www.axelos.com/gempdf/Axelos_Common_Glossary_2013.pdf

[12] E. Evans, "Eric evans blog - nosql," http://blog.sym-link.com/2009/05/12/nosql_2009.html, Accessed: 2014-05-01.

[13] R. Lamllari, "Extending a methodology for migration of the database layer to the cloud considering relational database schema migration to nosql," Master's thesis, University of Stuttgart, 2013.

[14] F. Leal, "Systematic mapping spreadsheet," https://docs.google.com/spreadsheets/d/1N3DboEqthdiKG3VDMKlqyjFdHC3pxrN1XtXtNoNrhX8, Accessed: 2015-05-01.

[15] A. Tahir and S. Macdonell, "A systematic mapping study on dynamic metrics and software quality," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, Sept 2012, pp. 326–335.

[16] D. Ameller, X. Burgus, O. Collell, D. Costal, X. Franch, and M. P. Papazoglou, "Development of service-oriented architectures using model-driven development: A mapping study," *Information and Software Technology*, vol. 62, no. 0, pp. 42 – 66, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584915000361

[17] F. Leal, "Fabio leal - pdf tokens matcher," https://github.com/fabiosl/pdf-tokens-finder, Accessed: 2015-05-01.

[18] C.-W. Huang, W.-H. Hu, C.-C. Shih, B.-T. Lin, and C.-W. Cheng, "The improvement of auto-scaling mechanism for distributed database - a case study for mongodb," in *Network Operations and Management Symposium (APNOMS), 2013 15th Asia-Pacific*, Sept 2013, pp. 1–3.

[19] L. Zhao, S. Sakr, and A. Liu, "A framework for consumer-centric sla management of cloud-hosted databases," *Services Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.

[20] Z. Zheng, J. Zhu, and M. Lyu, "Service-generated big data and big data-as-a-service: An overview," in *Big Data (BigData Congress), 2013 IEEE International Congress on*, June 2013, pp. 403–410.

[21] R. Moussa, "Benchmarking data warehouse systems in the cloud," in *Computer Systems and Applications (AICCSA), 2013 ACS International Conference on*, May 2013, pp. 1–8.

[22] P. Xiong, "Dynamic management of resources and workloads for rdbms in cloud: A control-theoretic approach," in *Proceedings of the on SIGMOD/PODS 2012 PhD Symposium*, ser. PhD '12. New York, NY, USA: ACM, 2012, pp. 63–68. [Online]. Available: http://doi.acm.org/10.1145/2213598.2213614

[23] E. Alomari, A. Barnawi, and S. Sakr, "Cdport: A framework of data portability in cloud platforms," in *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, ser. iiWAS '14. New York, NY, USA: ACM, 2014, pp. 126–133. [Online]. Available: http://doi.acm.org/10.1145/2684200.2684324

[24] C. Pahl and H. Xiong, "Migration to paas clouds - migration process and architectural concerns," in *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2013 IEEE 7th International Symposium on the*, Sept 2013, pp. 86–91.

[25] L. Zhao, S. Sakr, and A. Liu, "Application-managed replication controller for cloud-hosted databases," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, June 2012, pp. 922–929.

[26] S. Sakr, "Cloud-hosted databases: technologies, challenges and opportunities," *Cluster Computing*, vol. 17, no. 2, pp. 487–502, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10586-013-0290-7

[27] S. Sakr and A. Liu, "Is your cloud-hosted database truly elastic?" in *Services (SERVICES), 2013 IEEE Ninth World Congress on*, June 2013, pp. 444–447.

[28] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to adapt applications for the cloud environment," *Computing*, vol. 95, no. 6, pp. 493–535, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00607-012-0248-2

[29] E. Boytsov, "Designing and development of an imitation model of a multitenant database cluster," *Automatic Control and Computer Sciences*, vol. 48, no. 7, pp. 437–444, 2014. [Online]. Available: http://dx.doi.org/10.3103/S0146411614070049

[30] B. Sodhi and T. Prabhakar, "Assessing suitability of cloud oriented platforms for application development," in *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*, June 2011, pp. 328–335.

[31] D. Petcu, G. Macariu, S. Panica, and C. Crciun, "Portable cloud applicationsfrom theory to practice," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1417 – 1430, 2013, including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for {P2P} Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12000210

[32] G. Giannikis, D. Makreshanski, G. Alonso, and D. Kossmann, "Workload optimization using shareddb," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 1045–1048. [Online]. Available: http://doi.acm.org/10.1145/2463676.2463678

[33] K. Grolinger, W. Higashino, A. Tiwari, and M. Capretz, "Data management in cloud environments: Nosql and newsql data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, p. 22, 2013. [Online]. Available: http://www.journalofcloudcomputing.com/content/2/1/22

[34] A. Copie, T.-F. Fortis, V. Munteanu, and V. Negru, "Service datastores in cloud governance," in *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, July 2012, pp. 473–478.

[35] A. Copie, T.-F. Fortis, and V. Munteanu, "Determining the performance of the databases in the context of cloud governance," in *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, Oct 2013, pp. 227–234.

[36] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. HacigümüŞ, "Activesla: A profit-oriented admission control framework for database-as-a-service providers," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 15:1–15:14. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038931

[37] S. Srirama and A. Ostovar, "Optimal resource provisioning for scaling enterprise applications on the cloud," in *Cloud Computing Technology*

and Science (CloudCom), 2014 IEEE 6th International Conference on, Dec 2014, pp. 262–271.

[38] M. Dayarathna and T. Suzumura, "Towards scalable distributed graph database engine for hybrid clouds," in *Data-Intensive Computing in the Clouds (DataCloud), 2014 5th International Workshop on*, Nov 2014, pp. 1–8.

[39] L. Qiao, K. Surlaker, S. Das, T. Quiggle, B. Schulman, B. Ghosh, A. Curtis, O. Seeliger, Z. Zhang, A. Auradar, C. Beaver, G. Brandt, M. Gandhi, K. Gopalakrishna, W. Ip, S. Jgadish, S. Lu, A. Pachev, A. Ramesh, A. Sebastian, R. Shanbhag, S. Subramaniam, Y. Sun, S. Topiwala, C. Tran, J. Westerman, and D. Zhang, "On brewing fresh espresso: Linkedin's distributed data serving platform," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 1135–1146. [Online]. Available: http://doi.acm.org/10.1145/2463676.2465298

[40] S. Sakr, A. Liu, D. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *Communications Surveys Tutorials, IEEE*, vol. 13, no. 3, pp. 311–336, Third 2011.

[41] J. Montes, A. Snchez, B. Memishi, M. S. Prez, and G. Antoniu, "Gmone: A complete approach to cloud monitoring," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2026 – 2040, 2013, including Special sections: Advanced Cloud Monitoring Systems & The fourth {IEEE} International Conference on e-Science 2011 e-Science Applications and Tools & Cluster, Grid, and Cloud Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000496

[42] A. J. Elmore, S. Das, A. Pucher, D. Agrawal, A. El Abbadi, and X. Yan, "Characterizing tenant behavior for placement and crisis mitigation in multitenant dbmss," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 517–528. [Online]. Available: http://doi.acm.org/10.1145/2463676.2465308

[43] M. Dayarathna and T. Suzumura, "Graph database benchmarking on cloud environments with xgdbench," *Automated Software Engineering*, vol. 21, no. 4, pp. 509–533, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10515-013-0138-7

[44] H. Hu, Y. Wen, T.-S. Chua, and X. Li, "Toward scalable systems for big data analytics: A technology tutorial," *Access, IEEE*, vol. 2, pp. 652–687, 2014.

[45] D. Shue and M. J. Freedman, "From application requests to virtual iops: Provisioned key-value storage with libra," in *Proceedings of the Ninth European Conference on Computer Systems*, ser. EuroSys '14. New York, NY, USA: ACM, 2014, pp. 17:1–17:14. [Online]. Available: http://doi.acm.org/10.1145/2592798.2592823

[46] A. Schram and K. M. Anderson, "Mysql to nosql: Data modeling challenges in supporting scalability," in *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, ser. SPLASH '12. New York, NY, USA: ACM, 2012, pp. 191–202. [Online]. Available: http://doi.acm.org/10.1145/2384716.2384773

[47] C. BĂZĂR, C. S. IOSIF *et al.*, "The transition from rdbms to nosql. a comparative analysis of three popular non-relational solutions: Cassandra, mongodb and couchbase," *Database Systems Journal*, vol. 5, no. 2, pp. 49–59, 2014.

[48] A. Gomez, R. Ouanouki, A. April, and A. Abran, "Building an experiment baseline in migration process from sql databases to column oriented no-sql databases," *J Inform Tech Softw Eng*, vol. 4, no. 137, p. 2, 2014.

[49] Y. Kouki, F. de Oliveira, S. Dupont, and T. Ledoux, "A language support for cloud elasticity management," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, May 2014, pp. 206–215.

[50] "Full db json," https://gist.github.com/fabiosl/18b5e826c9daebda5165, Accessed: 2015-04-21.

[51] R. Cattell, "Scalable sql and nosql data stores," *SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1978915.1978919

[52] C. Mohan, "History repeats itself: Sensible and nonsensql aspects of the nosql hoopla," in *Proceedings of the 16th International Conference on Extending Database Technology*, ser. EDBT '13. New York, NY, USA: ACM, 2013, pp. 11–16. [Online]. Available: http://doi.acm.org/10.1145/2452376.2452378

[53] I. Konstantinou, E. Angelou, D. Tsoumakos, C. Boumpouka, N. Koziris, and S. Sioutas, "Tiramola: Elastic nosql provisioning through a cloud management platform," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 725–728. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213943

[54] M. Klems, D. Bermbach, and R. Weinert, "A runtime quality measurement framework for cloud database service systems," in *Proceedings of the 2012 Eighth International Conference on the Quality of Information and Communications Technology*, ser. QUATIC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 38–46. [Online]. Available: http://dx.doi.org/10.1109/QUATIC.2012.17

[55] ——, "A runtime quality measurement framework for cloud database service systems," in *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*, Sept 2012, pp. 38–46.

[56] "New relic," http://newrelic.com, Accessed: 2014-05-01.

[57] "Appsee," https://www.appsee.com/, Accessed: 2014-05-01.

[58] "Datadog," https://www.datadoghq.com/, Accessed: 2014-05-01.

[59] Y. Kouki and T. Ledoux, "CSLA : a Language for improving Cloud SLA Management," in *International Conference on Cloud Computing and Services Science, CLOSER 2012*, Porto, Portugal, Apr. 2012, pp. 586–591. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00675077

[60] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web Services Agreement Specification (WS-Agreement)," Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, Tech. Rep., Sep. 2005.

[61] S. Nepal, J. Zic, and S. Chen, "Wsla+: Web service level agreement language for collaborations," in *Services Computing, 2008. SCC '08. IEEE International Conference on*, vol. 2, July 2008, pp. 485–488.

[62] V. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low level metrics to high level slas - lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments," in *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, June 2010, pp. 48–54.

APÊNDICE A – Execution Reports

Execution Report 01

```
Running with Dataset Size: 100
...
MySQL Queries: [(50, 0.1901998519897461), (50, 0.1845710277557373), (50,
    0.18726801872253418), (30, 0.18717598915100098), (30, 0.1875), (30,
    0.18777203559875488), (20, 0.18753886222839355), (20, 0.18907999992370605),
    (20, 0.18427801132202148), (10, 0.1874251365661621), (10,
    0.18940091133117676), (10, 0.18792486190795898), (5, 0.18765616416931152),
    (5, 0.1839289665222168), (5, 0.18367600440979004), (2, 0.1840989589691162),
    (2, 0.18876910209655762), (2, 0.18515491485595703)]
MySQL SLA is broken? False

########################################################################

Running with Dataset Size: 1000
...
MySQL Queries: [(50, 0.20785880088806152), (50, 0.1742258071899414), (50,
    0.17499303817749023), (30, 0.1735529899597168), (30, 0.1874690055847168),
    (30, 0.17806601524353027), (20, 0.17487502098083496), (20,
    0.1774768829345703), (20, 0.17374396324157715), (10, 0.1745619773864746),
    (10, 0.17732787132263184), (10, 0.17468595504760742), (5,
    0.17685604095458984), (5, 0.17804884910583496), (5, 0.1777651309967041),
    (2, 0.17557621002197266), (2, 0.17390799522399902), (2,
    0.17412090301513672)]
MySQL SLA is broken? False

########################################################################
```

```
Running with Dataset Size: 10000

MySQL Queries: [(50, 0.2052299976348877), (50, 0.19243288040161133), (50,
    0.19191193580627441), (30, 0.19255805015563965), (30, 0.1887669563293457),
    (30, 0.19298601150512695), (20, 0.1880810260772705), (20,
    0.19396185874938965), (20, 0.19248104095458984), (10, 0.18845915794372559),
    (10, 0.19563794136047363), (10, 0.19304609298706055), (5,
    0.18861603736877441), (5, 0.1923990249633789), (5, 0.18983006477355957),
    (2, 0.1927800178527832), (2, 0.18825697898864746), (2, 0.18577885627746582)]
MySQL SLA is broken? False

########################################################################

Running with Dataset Size: 100000
...
MySQL Queries: [(50, 0.5140020847320557), (50, 0.30184102058410645), (50,
    0.3160550594329834), (30, 0.30850696563720703), (30, 0.3058919906616211),
    (30, 0.31014108657836914), (20, 0.33054089546203613), (20,
    0.34288787841796875), (20, 0.3059229850769043), (10, 0.30807995796203613),
    (10, 0.33196115493774414), (10, 0.30630993843078613), (5,
    0.30664610862731934), (5, 0.30666184425354004), (5, 0.3065659999847412),
    (2, 0.30725812911987305), (2, 0.3074190616607666), (2, 0.33382105827331543)]
MySQL SLA is broken? False

########################################################################

Running with Dataset Size: 1000000
...

MySQL Queries: [(50, 2.9074718952178955), (50, 3.169628143310547), (50,
    3.068336009979248), (30, 2.946420907974243), (30, 2.983638048171997), (30,
    3.0676960945129395), (20, 3.0680489540100098), (20, 3.0676097869873047),
    (20, 3.272584915161133), (10, 3.0661330223083496), (10,
    2.9682278633117676), (10, 3.0668060779571533), (5, 3.0365960597991943), (5,
    3.0023999214172363), (5, 3.267791986465454), (2, 2.9456560611724854), (2,
    2.9858040809631348), (2, 3.067668914794922)]
MySQL SLA is broken? True
```

```
########################################################################

Running with Dataset Size: 2000000
...
MySQL Queries: [(50, 5.984281063079834), (50, 5.900359869003296), (50,
    5.8593878746032715), (30, 5.726871013641357), (30, 6.033170938491821), (30,
    5.932153940200806), (20, 5.8248679637908936), (20, 5.867033004760742), (20,
    6.104353904724121), (10, 6.338104963302612), (10, 6.007672071456909), (10,
    5.742549896240234), (5, 5.86237096786499), (5, 5.806718111038208), (5,
    6.033854007720947), (2, 6.1649181842803955), (2, 6.007449150085449), (2,
    5.827448844909668)]
MySQL SLA is broken? True

########################################################################

Running with Dataset Size: 3000000
...
MySQL Queries: [(50, 12.227512836456299), (50, 8.691967964172363), (50,
    9.1024010181427), (30, 8.589695930480957), (30, 8.678112030029297), (30,
    8.629335165023804), (20, 8.566416025161743), (20, 8.55915904045105), (20,
    8.624006986618042), (10, 8.615052938461304), (10, 9.280121088027954), (10,
    8.590642213821411), (5, 8.700731992721558), (5, 8.774307012557983), (5,
    8.706260919570923), (2, 8.449976921081543), (2, 8.513587951660156), (2,
    8.501986026763916)]
MySQL SLA is broken? True
```

Execution Report 02

```
Elastic Queries: [(50, 0.37639307975769043), (50, 0.34427809715270996), (50,
    0.3618030548095703), (30, 0.35060906410217285), (30, 0.3810849189758301),
    (30, 0.35098695755004883), (20, 0.35475611686706543), (20,
    0.36417102813720703), (20, 0.37682604789733887), (10, 0.3461129665374756),
    (10, 0.36162900924682617), (10, 0.3642888069152832), (5,
    0.3701000213623047), (5, 0.3491690158843994), (5, 0.36877894401550293), (2,
    0.3788158893585205), (2, 0.38399386405944824), (2, 0.3639249801635742)]
Elastic SLA is broken? False
```

APÊNDICE A – Execution Reports

Execution Report 02 - Time in Millis

	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7	Exp8	Exp9	Exp10
1	58	58	40	39	30	32	28	24	28	26
2	65	55	55	41	31	34	22	31	24	19
3	59	50	47	33	40	37	29	29	26	17
4	52	57	52	40	43	41	28	25	26	22
5	50	58	43	49	38	36	31	27	29	20
6	54	57	51	40	38	36	26	26	26	17
7	55	50	46	43	37	35	24	26	21	18
8	61	50	40	44	37	33	24	28	22	22
9	52	47	42	46	40	35	25	22	21	22
10	55	61	44	38	34	32	26	24	19	29
11	58	58	42	44	36	33	24	22	31	22
12	57	56	42	40	33	32	23	32	26	20
13	53	51	45	42	35	35	27	20	15	19
14	60	50	48	49	38	37	29	26	20	24
15	61	47	44	42	36	33	31	23	24	24
16	55	45	40	39	33	35	27	22	26	22
17	56	47	44	42	24	31	26	25	19	18
18	54	49	44	41	31	24	26	25	20	21
19	57	56	39	43	31	30	29	20	19	18
20	51	47	46	39	43	30	24	19	18	20
21	53	49	42	44	41	29	26	26	18	15
22	56	47	49	38	36	35	28	23	16	14
23	51	51	40	45	39	31	22	20	16	19
24	56	44	51	39	34	32	30	28	21	21
25	54	50	43	40	41	33	23	30	18	15
26	55	45	42	44	32	32	29	29	24	21
27	51	48	52	39	35	28	31	29	23	23

28 51 62 46 39 35 29 33 28 30 24
29 56 55 42 36 31 37 26 22 106 20
30 56 52 50 37 34 36 30 25 26 25
31 58 58 40 43 31 32 23 30 25 23
32 56 59 44 36 38 27 26 32 21 18
33 51 52 39 48 30 32 27 27 25 19
34 47 48 38 41 34 35 24 28 23 18
35 49 49 41 45 32 35 27 21 24 20
36 48 54 39 44 37 24 29 20 22 23
37 47 57 45 37 38 26 30 18 21 22
38 51 49 42 46 31 24 25 25 23 24
39 52 48 41 38 30 29 28 28 23 20
40 59 47 49 32 33 27 25 24 21 22
41 59 50 41 47 34 28 28 27 23 20
42 53 53 42 37 36 30 22 29 18 25
43 50 47 44 39 38 28 29 23 19 19
44 55 53 39 38 32 32 23 26 21 23
45 55 54 40 34 35 27 27 21 16 29
46 51 45 47 37 33 29 28 24 23 20
47 54 47 37 35 42 33 27 25 22 16
48 53 58 41 41 35 33 23 24 21 25
49 54 48 48 44 35 27 32 27 24 23
50 53 41 47 38 34 31 20 25 20 17
51 57 47 46 38 32 35 27 23 23 25
52 53 49 47 32 39 35 29 22 16 21
53 51 49 38 37 40 31 22 19 30 18
54 56 43 41 37 30 30 28 24 32 20
55 54 51 51 31 37 33 33 24 15 22
56 49 47 40 43 38 33 25 22 19 17
57 51 45 41 42 30 37 24 26 19 16
58 50 44 35 41 31 27 27 24 24 17
59 49 43 47 33 34 29 29 22 21 17
60 58 52 51 42 36 31 23 22 19 18
61 58 49 45 36 26 36 27 26 16 17
62 57 43 45 32 37 32 29 23 24 20
63 60 51 39 46 30 34 25 20 18 22
64 66 52 35 46 31 25 29 24 21 19
65 57 47 45 35 33 34 27 24 21 23

66 56 49 49 35 35 28 27 26 19 16
67 55 46 38 38 31 29 28 24 21 18
68 52 51 38 40 32 38 28 20 20 20
69 53 50 42 43 29 28 26 20 19 24
70 48 50 43 38 35 28 21 24 21 23
71 51 44 37 38 32 34 20 22 19 13
72 54 49 43 36 39 31 25 27 19 22
73 49 51 50 38 31 28 31 25 24 17
74 47 42 36 40 32 28 26 30 18 18
75 49 48 45 41 38 27 29 21 18 23
76 54 46 45 38 33 33 32 22 18 19
77 50 49 35 42 33 28 26 26 24 19
78 54 49 45 42 36 29 27 27 20 19
79 52 45 44 40 36 28 28 22 21 19
80 50 44 47 41 33 24 35 27 23 18
81 54 43 44 50 40 21 26 27 23 21
82 49 43 40 35 26 34 28 22 24 17
83 53 40 40 44 30 37 31 25 22 19
84 48 52 41 38 27 29 24 27 19 17
85 51 52 43 38 33 27 25 31 17 20
86 49 44 46 31 40 26 24 25 17 17
87 55 43 35 38 31 31 26 22 24 18
88 60 46 38 42 25 29 27 26 24 16
89 50 49 44 32 31 27 30 24 14 20
90 54 52 36 36 32 28 30 19 20 20
91 64 45 41 31 27 31 25 28 19 19
92 59 45 44 33 38 35 22 22 17 18
93 57 49 40 31 30 28 33 25 22 18
94 54 43 42 47 34 30 27 23 20 15
95 55 41 45 33 30 23 24 28 23 15
96 48 45 44 34 34 25 28 21 20 16
97 52 45 40 31 31 31 31 29 20 17
98 49 39 41 42 27 21 25 24 16 17
99 43 42 41 36 31 25 25 25 22 19
100 46 41 42 37 36 30 26 18 15 19