Defesa de Mestrado PPGSC / DIMAP / UFRN

SLA-BASED GUIDELINES FOR DATABASE TRANSITIONING

FABIO LEAL

ADVISOR: MARTIN MUSICANTE

AGENDA

- INTRODUCTION
- THE SYSTEMATIC MAPPING
- PROBLEM
- SOLUTION
- CASE STUDY
- CONCLUSIONS

OUR WORK – BRIEF STORY

- New DB technologies over the last years.
- DB transitioning scenarios on the industry.
- How this process should be done?
- Our proposed solution:
    - Set of SLA-Based guidelines to assess and guide database transitioning scenarios.
    - Validation by case study

SOME CONCEPTS - CLOUD COMPUTING

- (STANOEVSKA-SLABEVA; WOZNIAK, 2009)
  - Cloud computing is a new computing paradigm
  - The main features of clouds are virtualization and scalability on demand
  - Infrastructure Resources (HW, Storage, Software) provided in a X-as-a Service manner
  - Cloud services are consumed either via Web browser or via a defined API

SOME CONCEPTS – THE TECHNOLOGICAL SHIFT

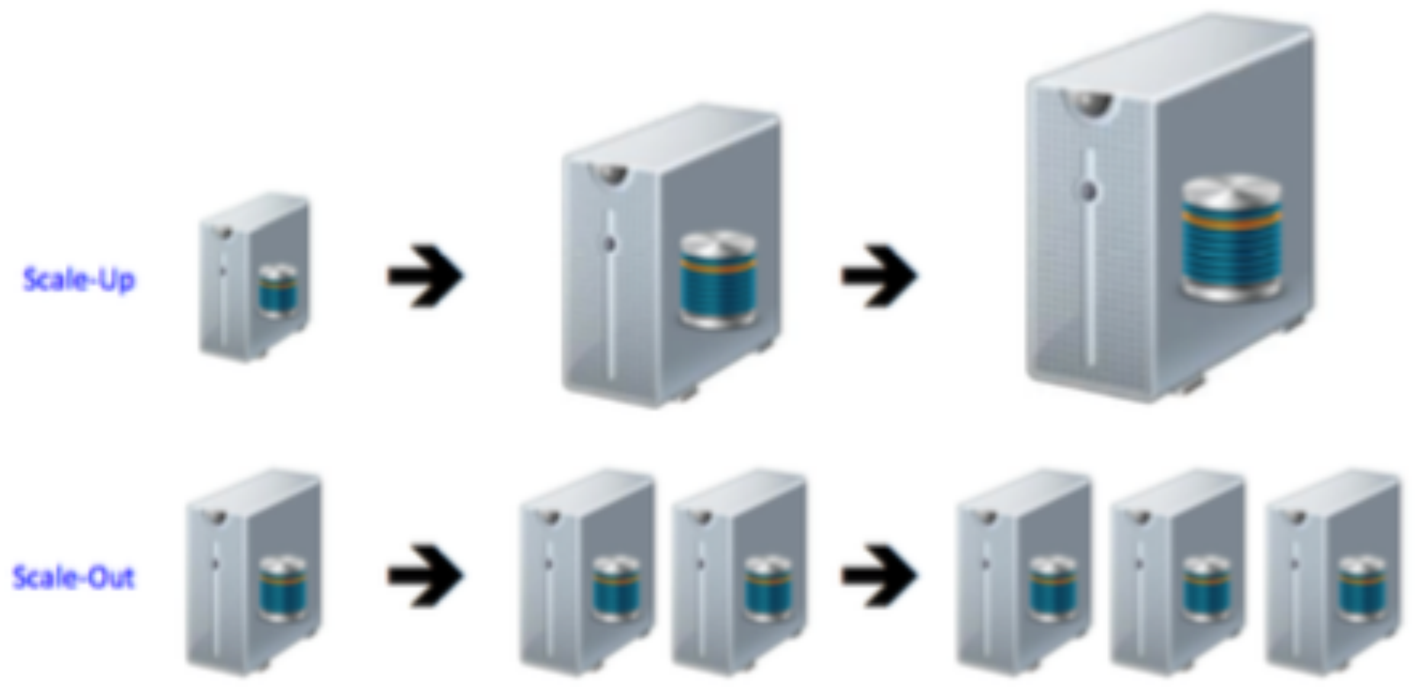


Figure 3: Scale Out vs Scale Up (DHANDALA, 2015).

DATA INTEGRATION, NOSQL MOVEMENT & POLYGLOT PERSISTENCE

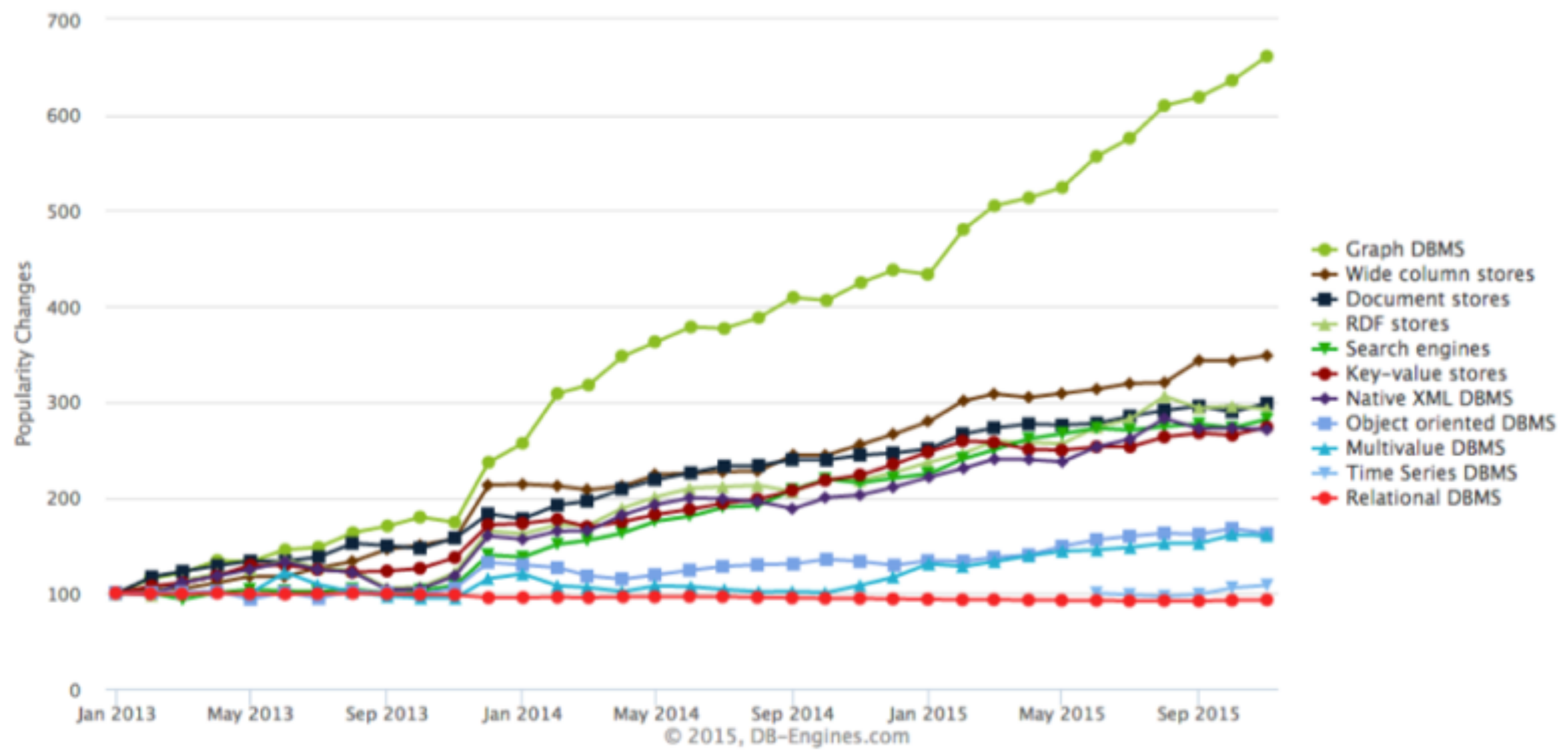


Figure 4: Database Popularity Growth Chart (RANKINGCHART, 2015).

DATA INTEGRATION, NOSQL MOVEMENT & POLYGLOT PERSISTENCE

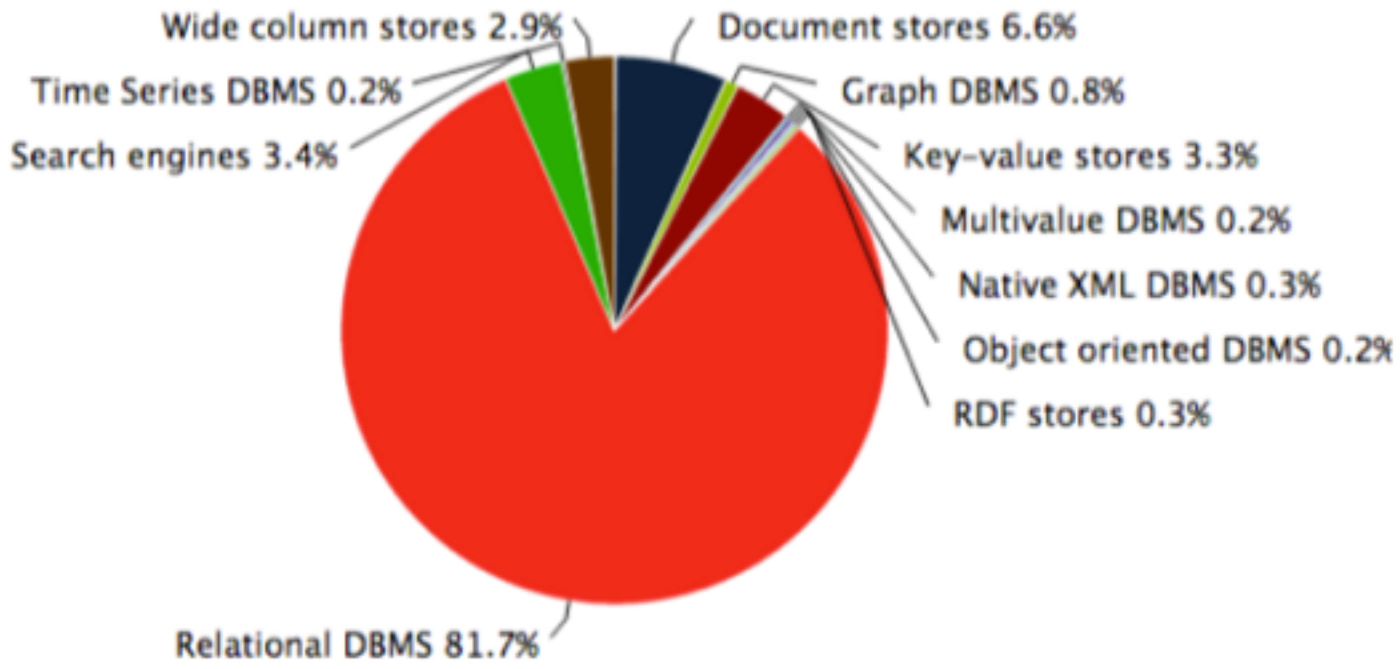
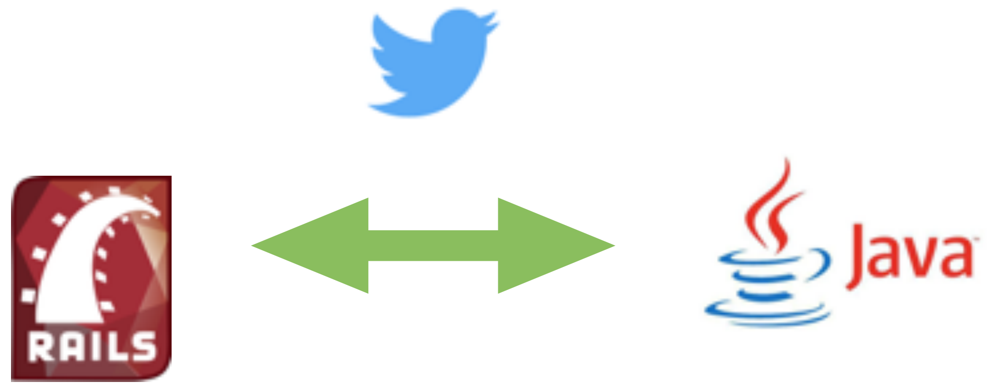
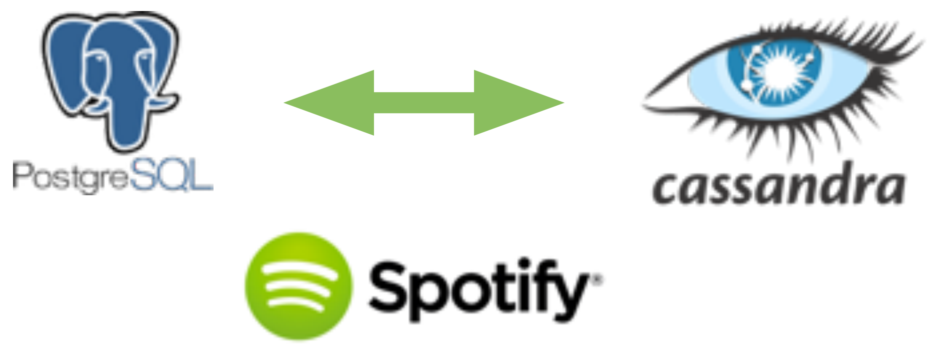


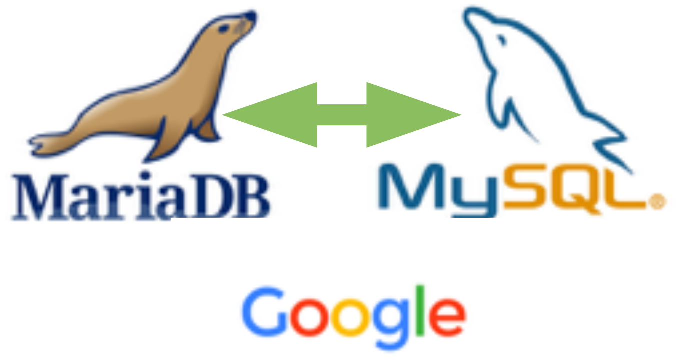
Figure 5: Database Popularity Chart - February/2016 (RANKINGCHART, 2015).

TRANSITIONING SCENARIOS



- What motivates?
  - Time To Market -> Not enough time for decisions;
  - Bad decisions, Scalability;
  - Any software component (Source code refactoring, database replacements);





SERVICE LEVEL AGREEMENT (SLA)

- *"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 re- sponsibilities of the IT service provider and the customer" (ITIL v3)*

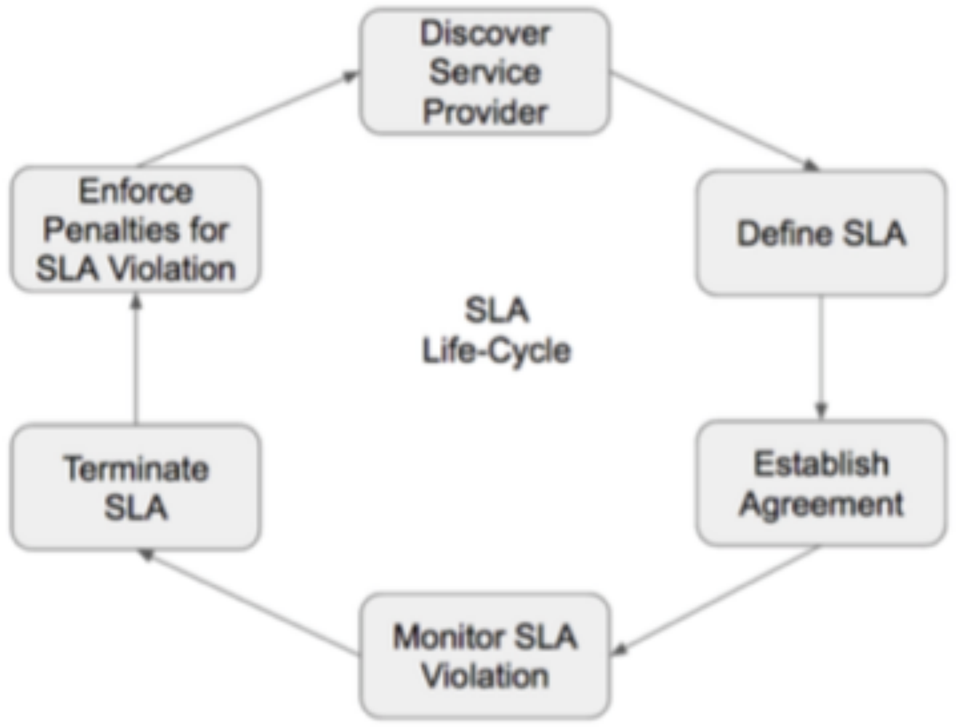


Figure 7: SLA Life-cycle (WU; BUYYA, 2012).

SYSTEMATIC MAPPINGS

- Answers research questions. Ex: *Cloud + The technological shift + NoSQL + Transitions: How database transitions should be done?*

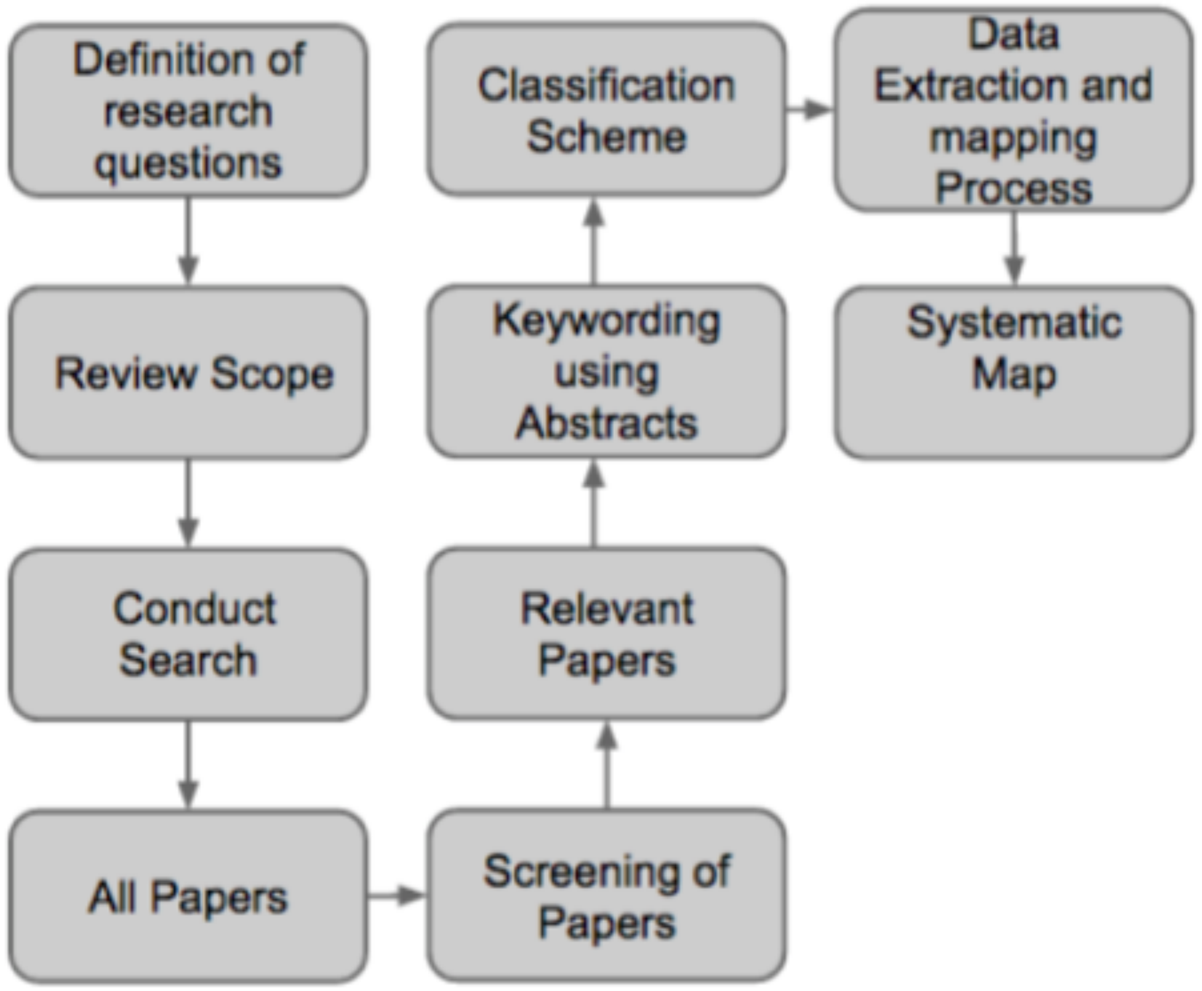


Figure 6: Systematic Mapping Steps (PETERSEN et al., 2008).

CONNECTING THE DOTS

- Cloud Computing
- The Technological Shift
- NoSQL & Polyglot Persistence
- Transitioning Scenarios
- Service Level Agreements
- **Systematic Mapping!**

SYSTEMATIC MAPPING QUESTIONS

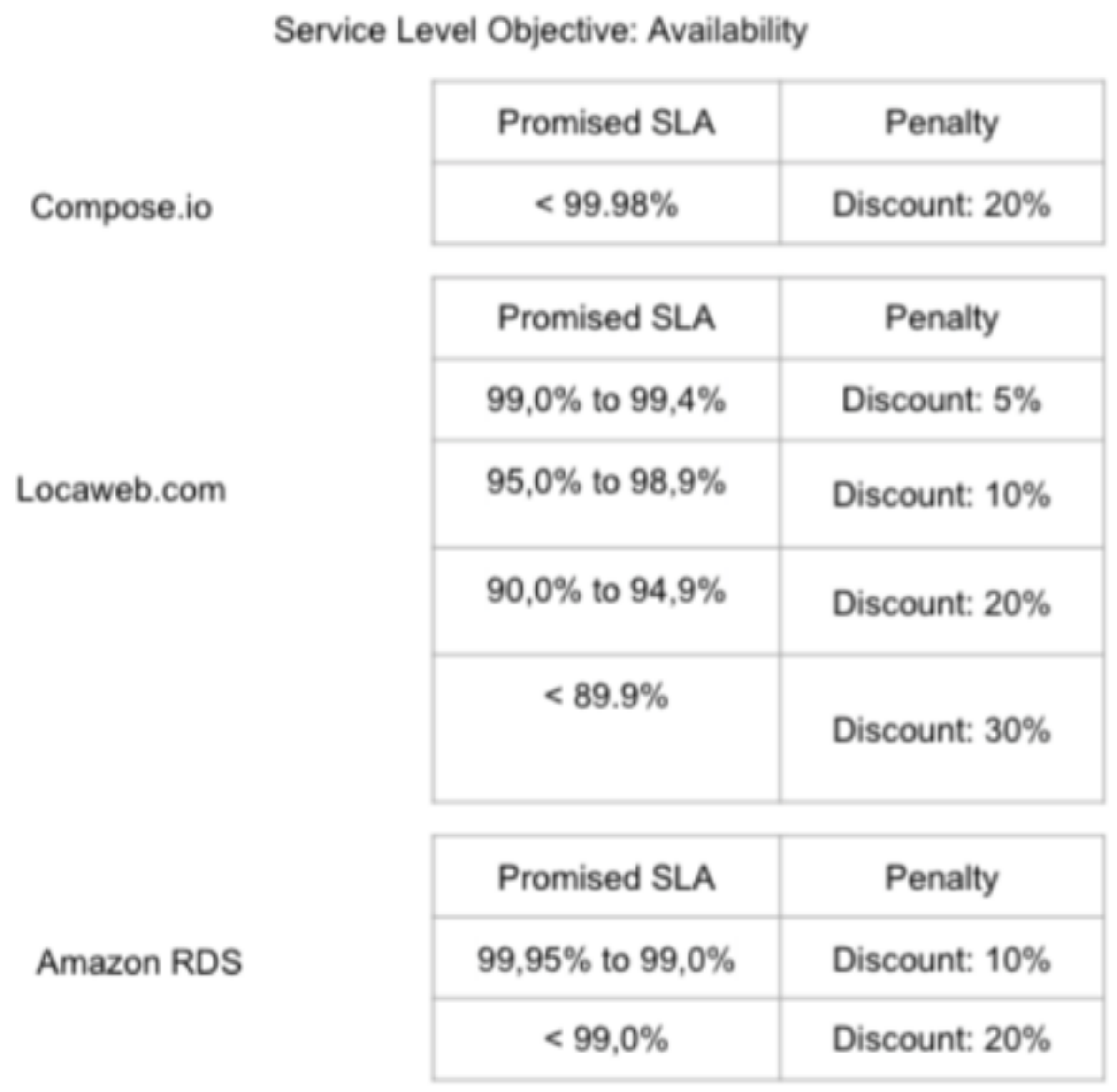
- RQ1) Reasons to change from RDBMSs to NoSQL solutions?
- AQ1.1) What are the pros and cons to migrate from RDBMSs to NoSQL solutions?
- AQ1.2) How can we measure the overall improvements promised by this change?
- RQ2) How can SLAs be used to guide database transitioning processes from RDBMSs to NoSQL databases in cloud-based apps?
- RQ3) Is there a standard representation of SLAs in cloud services?

SYSTEMATIC MAPPING OUTCOMES

- Over 70 publications analyzed
- Not many publications about database transitions (industry-related)
- Database transitions happen, most of the times, on the industry.
- No publication was found addresed the problem of measuring improvements **after a database transition**. TPC-H or any other benchmark process can be used.
- No standard way or process to transition databases emerged. Some industry reports were found - no standards.
- No standard representation of SLAs.

SERVICE LEVEL AGREEMENT (SLA)

- What?
  - Natural-language document
  - An ontology
  - An automated test suite
  - …



Service Level Objective: Availability

	Promised SLA	Penalty
Compose.io	< 99.98%	Discount: 20%
	Promised SLA	Penalty
Locaweb.com	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%
	Promised SLA	Penalty
Amazon RDS	99,95% to 99,0%	Discount: 10%
	< 99,0%	Discount: 20%

Figure 8: Service Level Objective: Availability on Cloud Services (LOCAWEBSLA, 2015)(COMPOSE.IO, 2015)(AMAZONRDS, 2015).

THE PROBLEM

- Using non-standardized methods in database transition scenarios can lead to non-desired scenarios (multiple transitions, bugs, etc.)
- Why and how database transitions should be done?
- 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?

THE SOLUTION

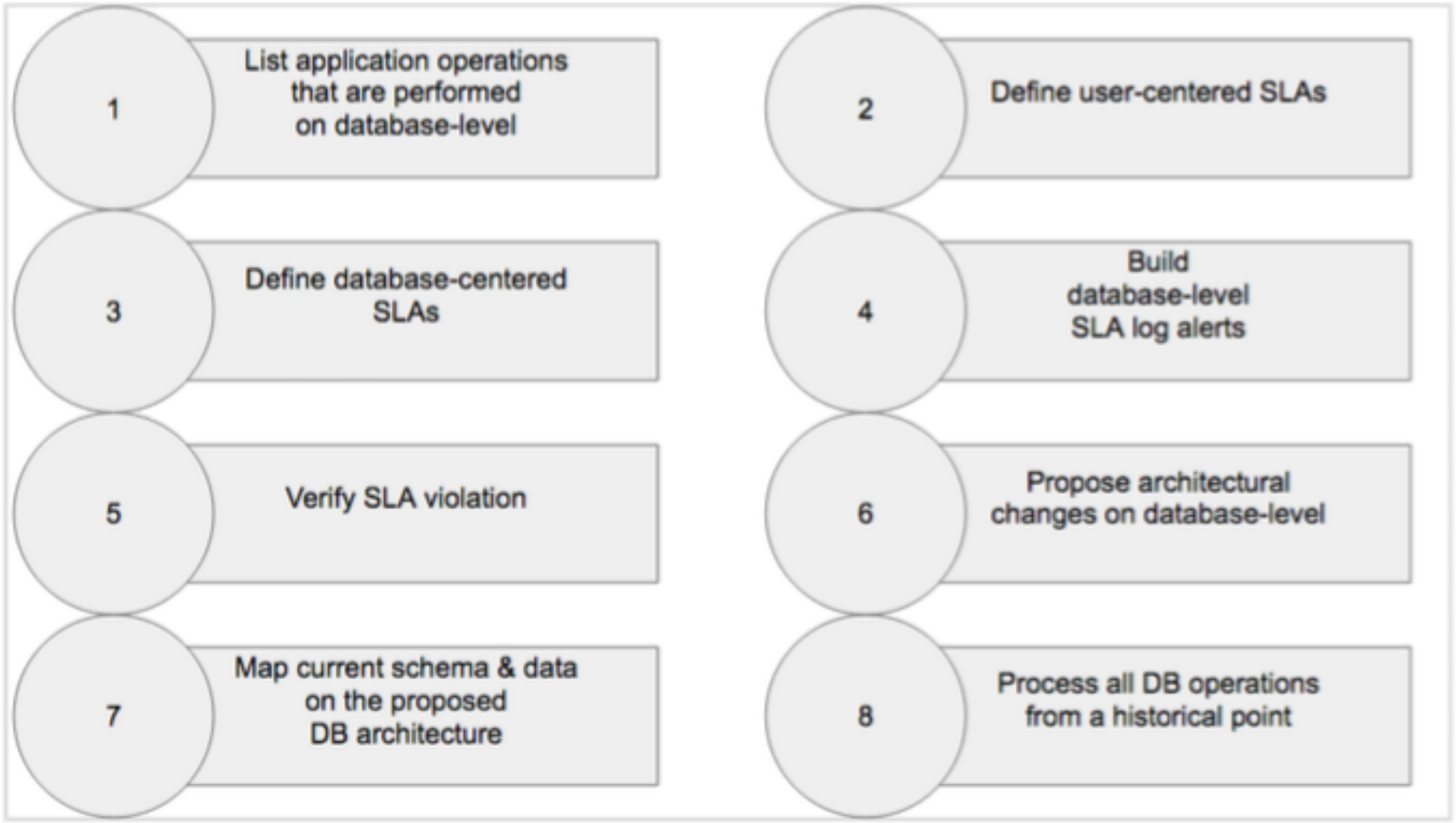

1
List application operations that are performed on database-level
2
Define user-centered SLAs
3
Define database-centered SLAs
4
Build database-level SLA log alerts
5
Verify SLA violation
6
Propose architectural changes on database-level
7
Map current schema & data on the proposed DB architecture
8
Process all DB operations from a historical point

Figure 9: Relational to NoSQL Steps.

1. LIST APPLICATION OPERATIONS THAT ARE PERFORMED AT DATABASE LEVEL

- Do not consider UI operations or business-logic-only operations

- On a Social Network, for instance
  - Follow or befriend another user;
  - Publish posts;
  - List user timeline;

2. DEFINE USER-CENTERED SLAS

- Ideal threshold
- Tolerable Threshold
- SLA-Delta Factor

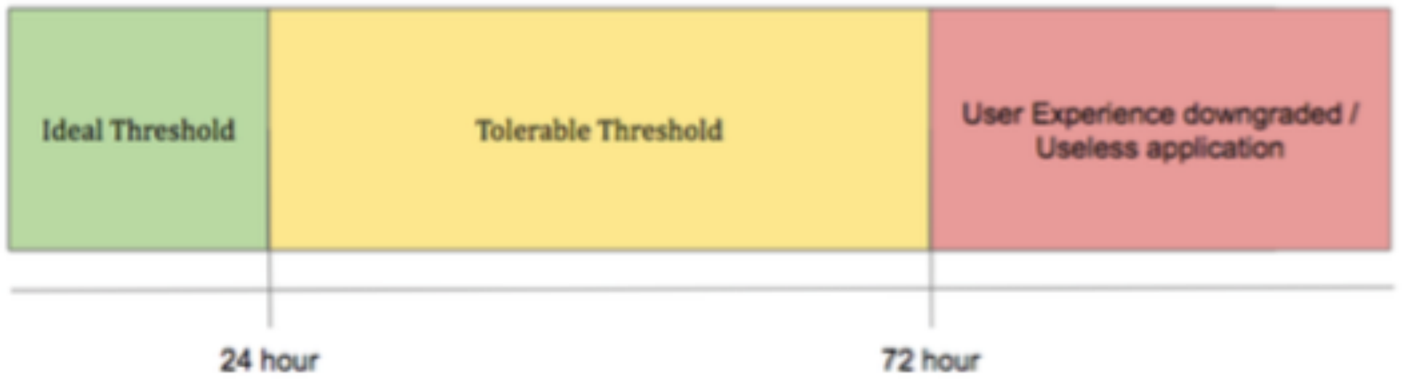
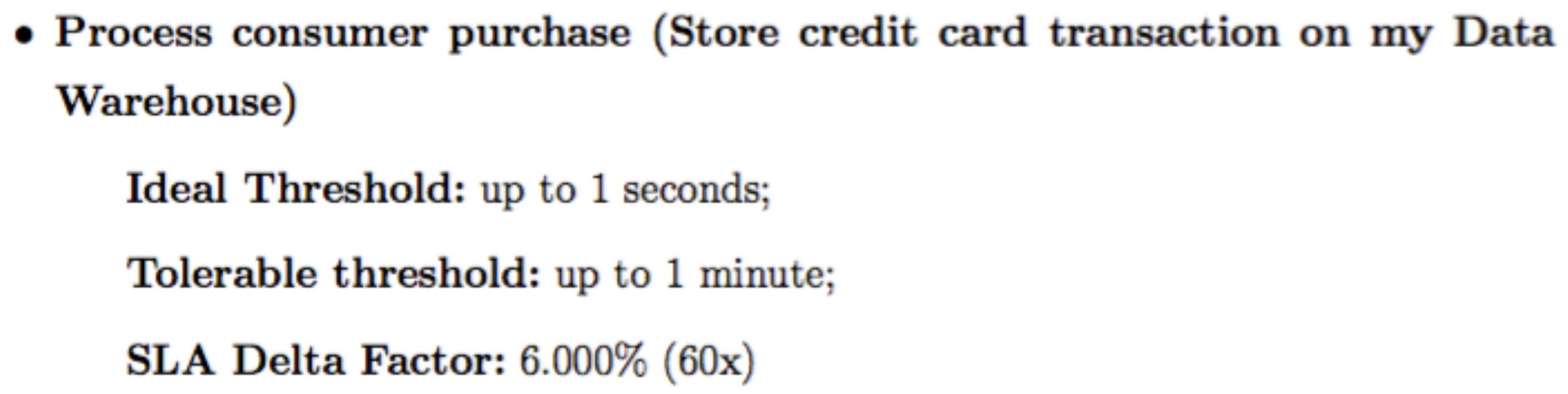


Figure 10: SLA Thresholds - 3x SLA Delta factor.



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

  Ideal Threshold: up to 1 seconds;

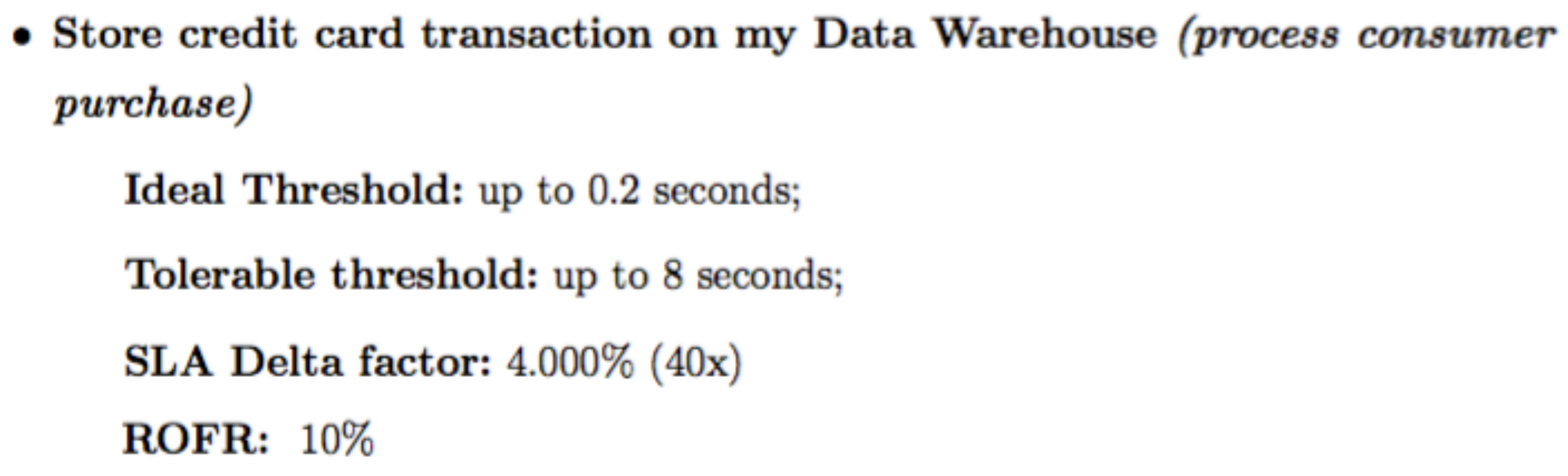
  Tolerable threshold: up to 1 minute;

  SLA Delta Factor: 6.000% (60x)

3. DEFINE DATABASE-CENTERED SLAS

- Ideal threshold
- Tolerable Threshold
- SLA-Delta Factor
- Rate of faulty requests

- Any *measurable metric (time, data freshness, CPU usage, etc)*



- **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 factor: 4.000% (40x)

  ROFR: 10%

4. BUILD DATABASE-LEVEL SLA LOG ALERTS

- 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.
- Log analyzers and alerts can be implemented within the source code of the application or using external services, such as (LOGSTASH, 2015), (PAPERTRAIL, 2015) and (NEWRELIC, 2014).
- Alerts contain the timestamp of when the alerts were fired

5. VERIFY SLA VIOLATION

- Mean number of operations increased?
- Hardware failure?
- New feature is demanding more DB resources?
- 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.
- *Point-in-time recovery*

6. PROPOSE ARCHITECTURAL CHANGES AT DATABASE LEVEL

- Add Database Indexes?
- Change the way how queries are done?
- Denormalizing tables help?
- Scale up?
- Switch Database?

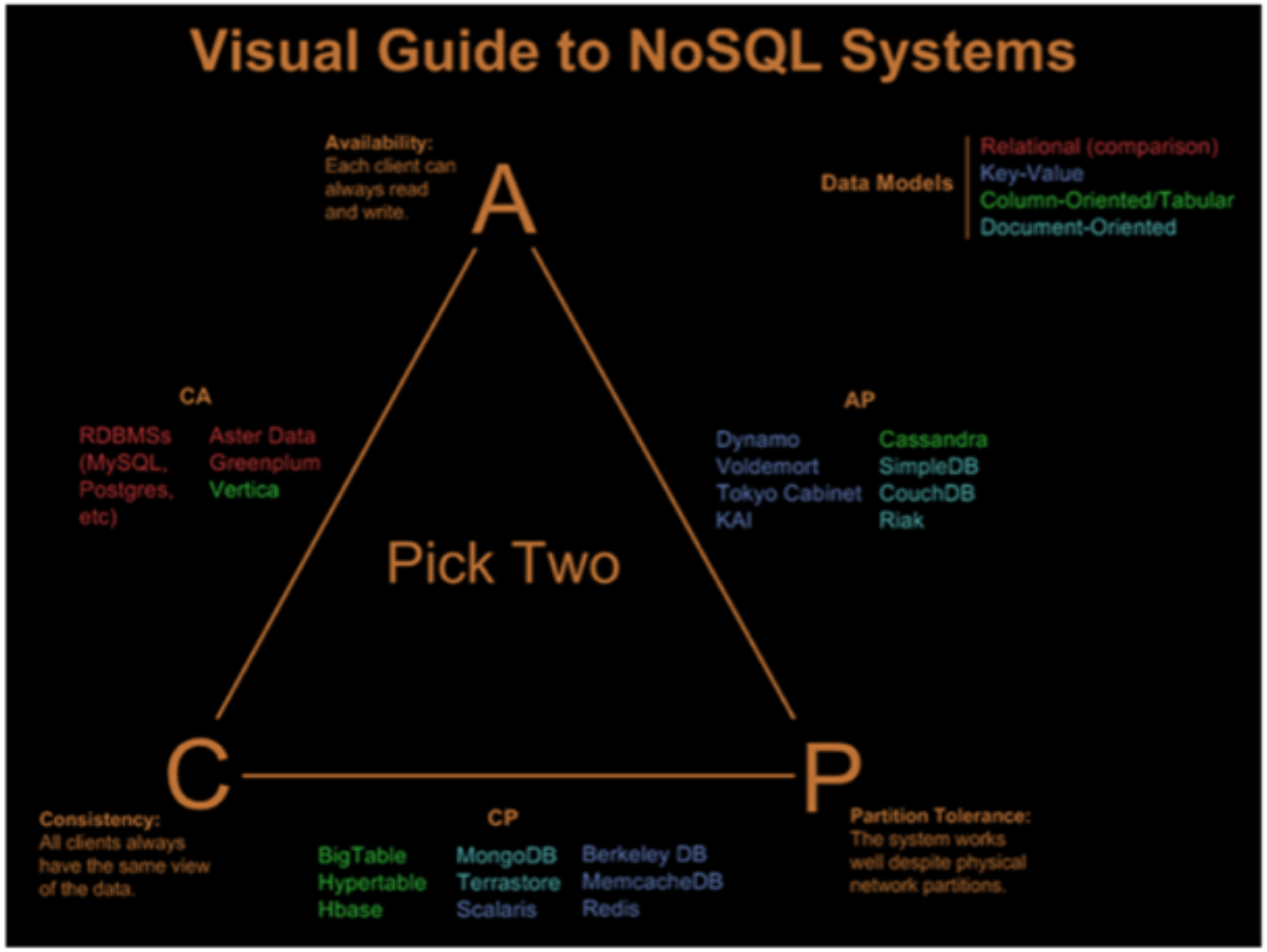
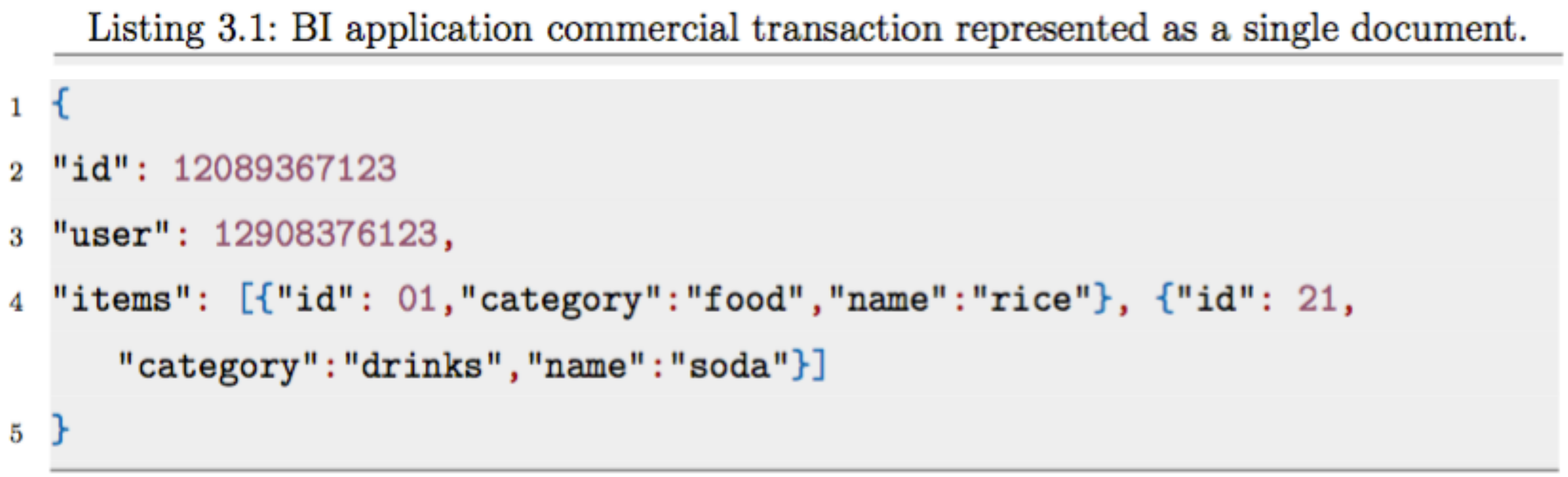


Figure 13: CAP Theorem (HAO, 2014a).

7. MAP CURRENT SCHEMA & DATA ON THE PROPOSED DATABASE ARCHITECTURE

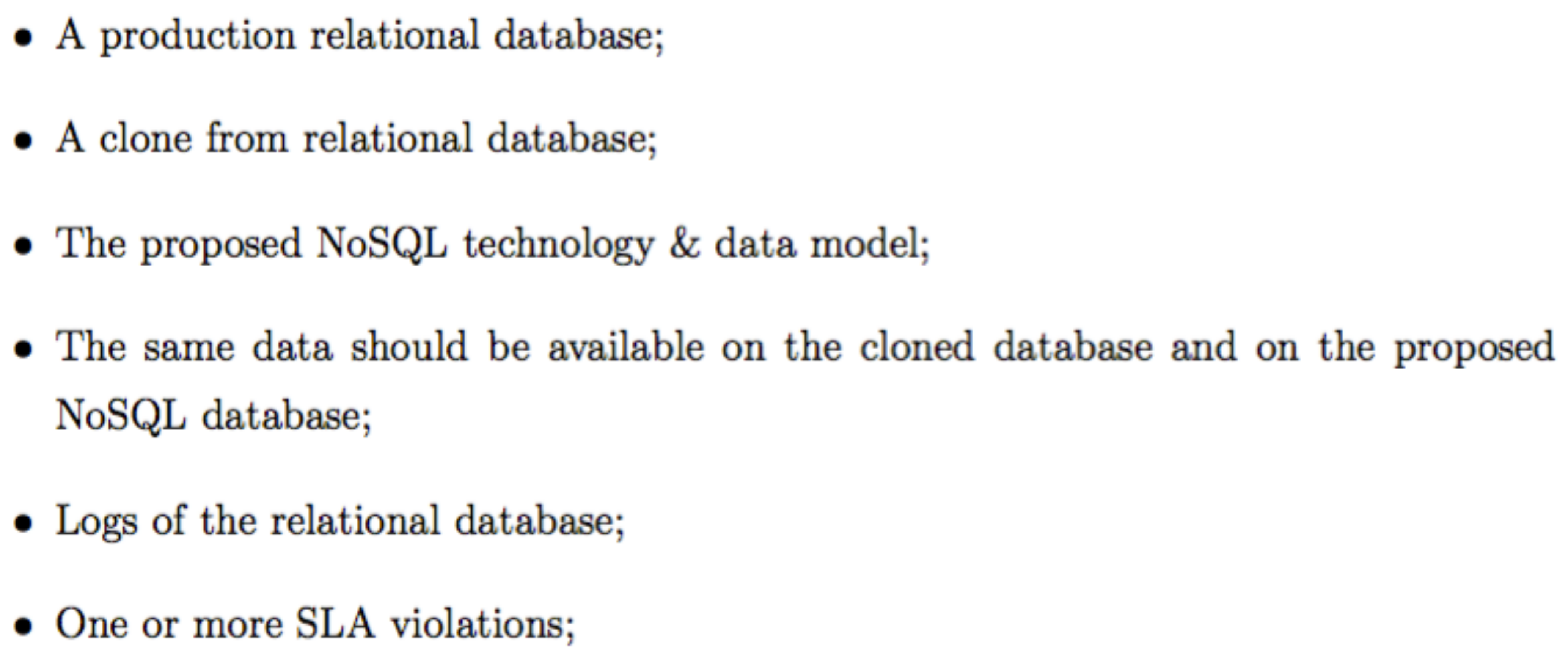
- Tables and relationships in a new data format
- Real data



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"}]
}
```

8. EXECUTE ALL OPERATIONS FROM A HISTORICAL POINT



- 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;

CASE STUDY

- First: Open Source Software (Wordpress, Redmine, Moodle, etc)
- Social Media Monitoring App

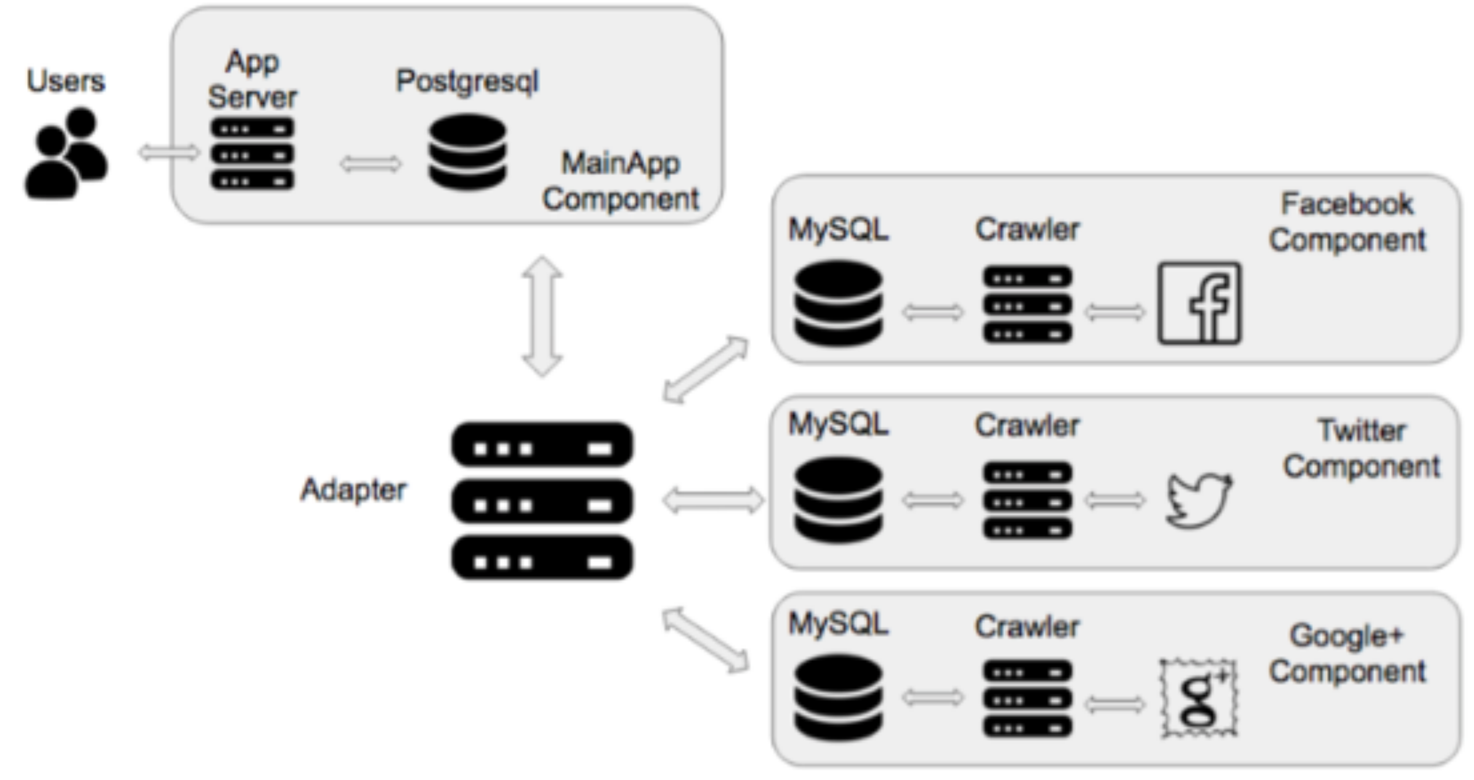
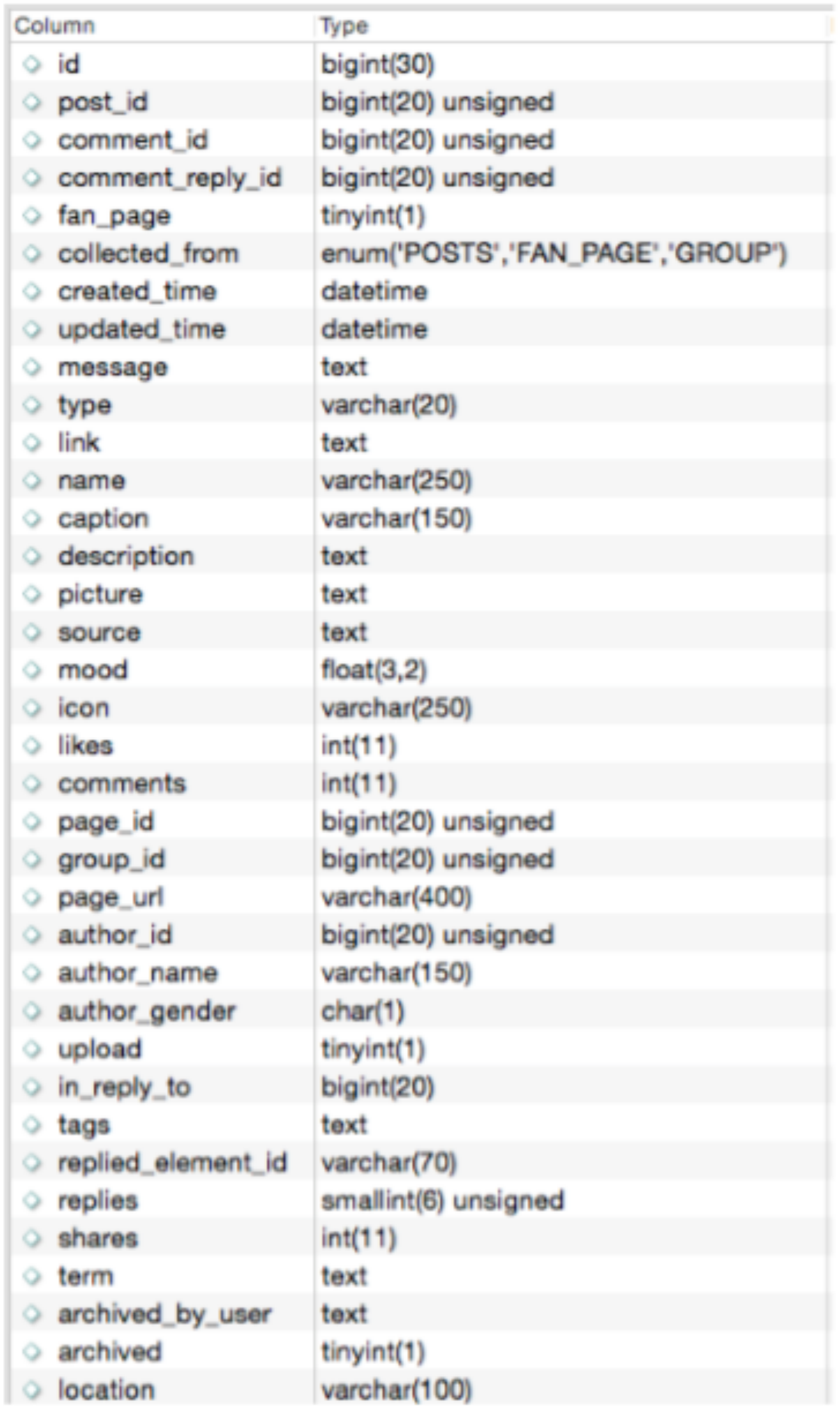


Figure 14: Proposed architecture - Social Media monitoring app.

CASE STUDY

- …Millions (or billions) of posts later…
  - Bad performance
    - Caused by data overload?
    - Should we use a better server?
    - Should we change our code architecture?
    - Should we use another database? - Silver Bullets Everywhere!
      - Why?
      - Prove it!



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)

Figure 15: Posts table.

1. LIST APPLICATION OPERATIONS THAT ARE PERFORMED AT DATABASE LEVEL

- Retrieve posts by ids
- Classify posts (add tags)
- Filter captured posts by filters

2. DEFINE USER-CENTERED SLAS

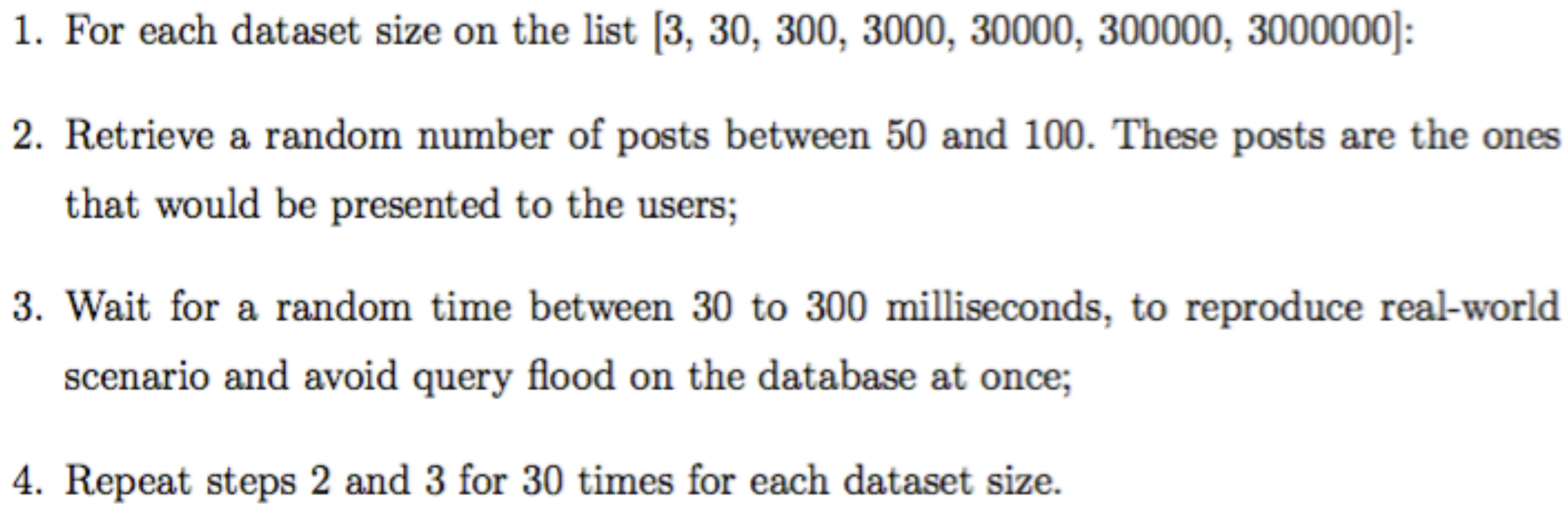
- Retrieve posts by ids:
  - Ideal threshold: 3 seconds
  - Tolerable threshold: 10 seconds
  - SLA Delta Factor: 3.3X
- Classify posts (add tags)
  - Ideal threshold: 1.5 second
  - Tolerable threshold: 3 second
  - SLA Delta Factor: 2x
- Filter Captured posts by Filters
  - Ideal threshold: 1.5 second
  - Tolerable threshold: 3 second
  - SLA Delta Factor: 2x

3. DEFINE DB-CENTERED SLAS

- Retrieve posts by ids:
  - Ideal threshold: 1 seconds
  - Tolerable threshold: 4 seconds
  - SLA Delta Factor: 4X
  - ROFR: 30%
- Filter Captured posts by Filters
  - Ideal threshold: 2 seconds
  - Tolerable threshold: 6 second
  - SLA Delta Factor: 3x
  - ROFR: 15%
- Classify posts (add tags)
  - Ideal threshold: 0.5 second
  - Tolerable threshold: 2 second
  - SLA Delta Factor: 4x
  - ROFR: 30%

4. BUILD DB-LEVEL SLA LOG ALERTS

- Operation 01:



1. For each dataset size on the list [3, 30, 300, 3000, 30000, 300000, 3000000]:
2. Retrieve a random number of posts between 50 and 100. These posts are the ones that would be presented to the users;
3. Wait for a random time between 30 to 300 milliseconds, to reproduce real-world scenario and avoid query flood on the database at once;
4. Repeat steps 2 and 3 for 30 times for each dataset size.

- Operations 02 and 03 follow the same pattern;
- Analyzers at source code level

5. VERIFY SLA VIOLATION

- Operation 01:
- No Issues
- The communication overhead (calls, loses importance as the number of DB records grow)

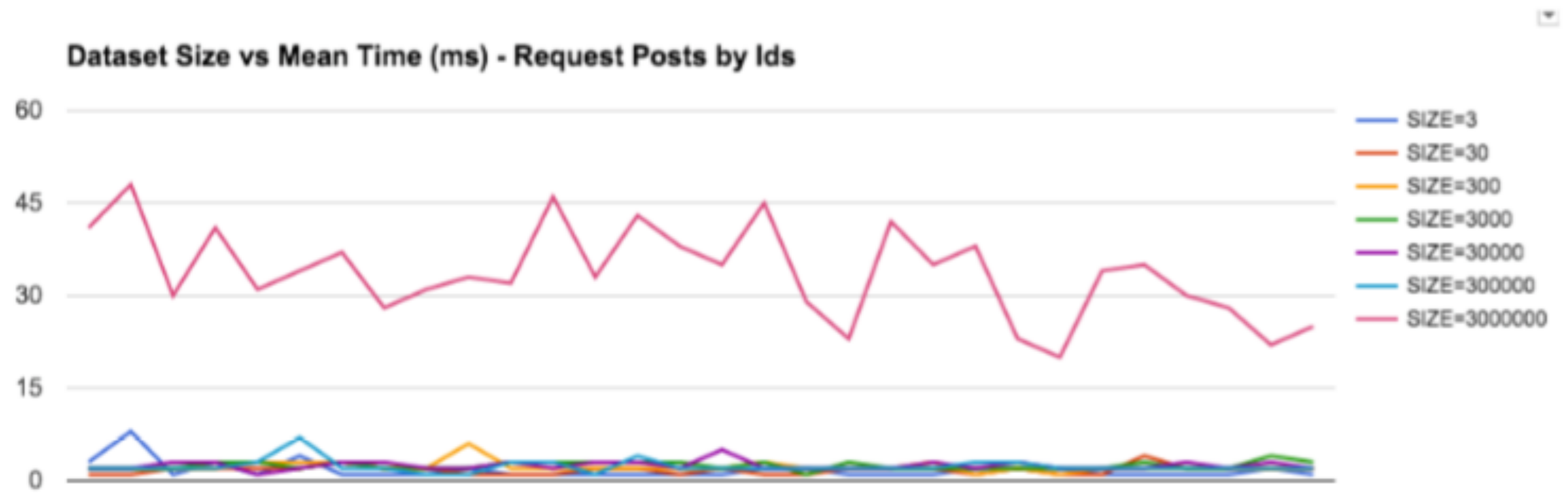


Figure 19: First Scenario - Retrieve post by ids.

5. VERIFY SLA VIOLATION

- Operation 02:
- SLA is broken on 300K and 3KK datasets

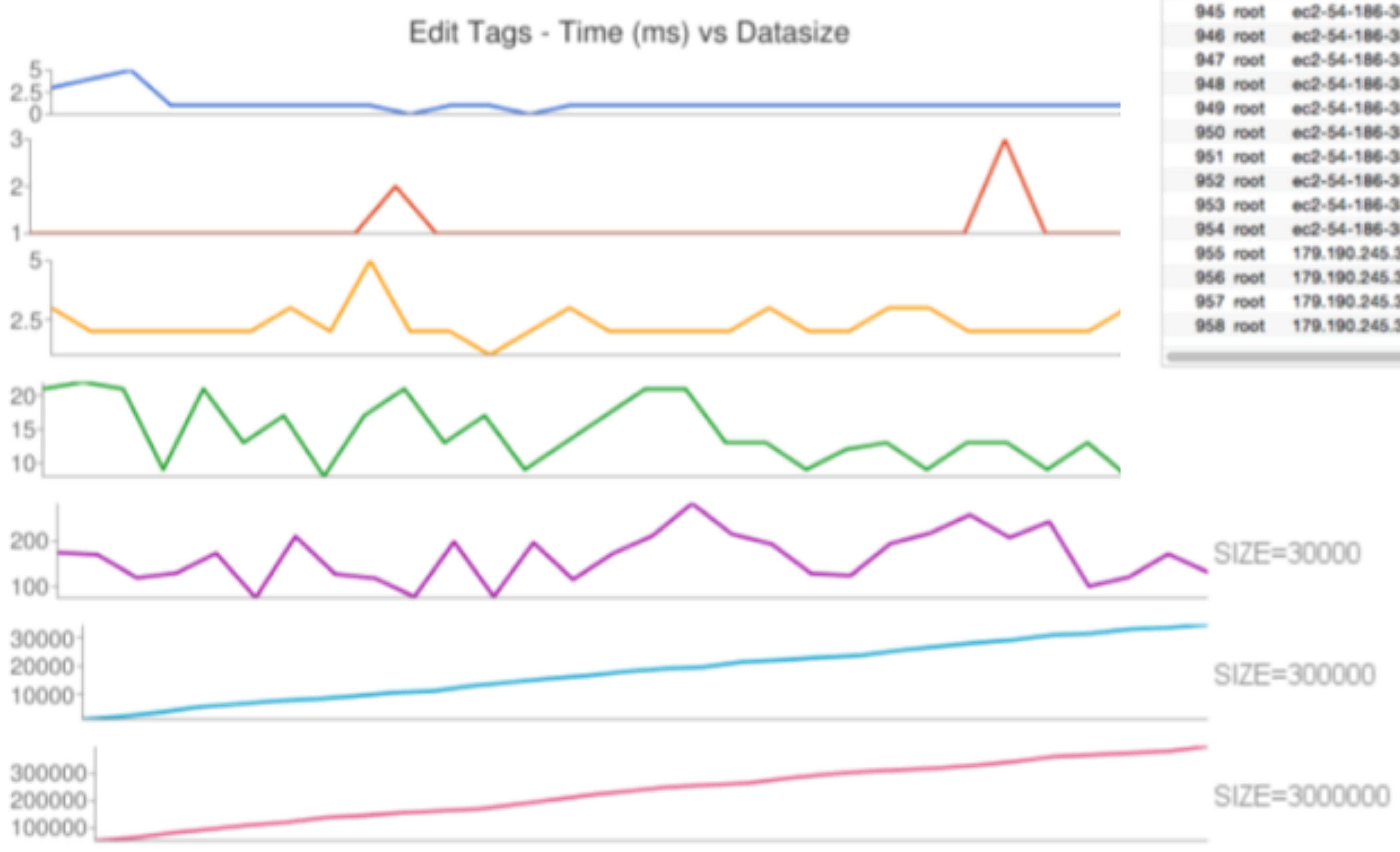
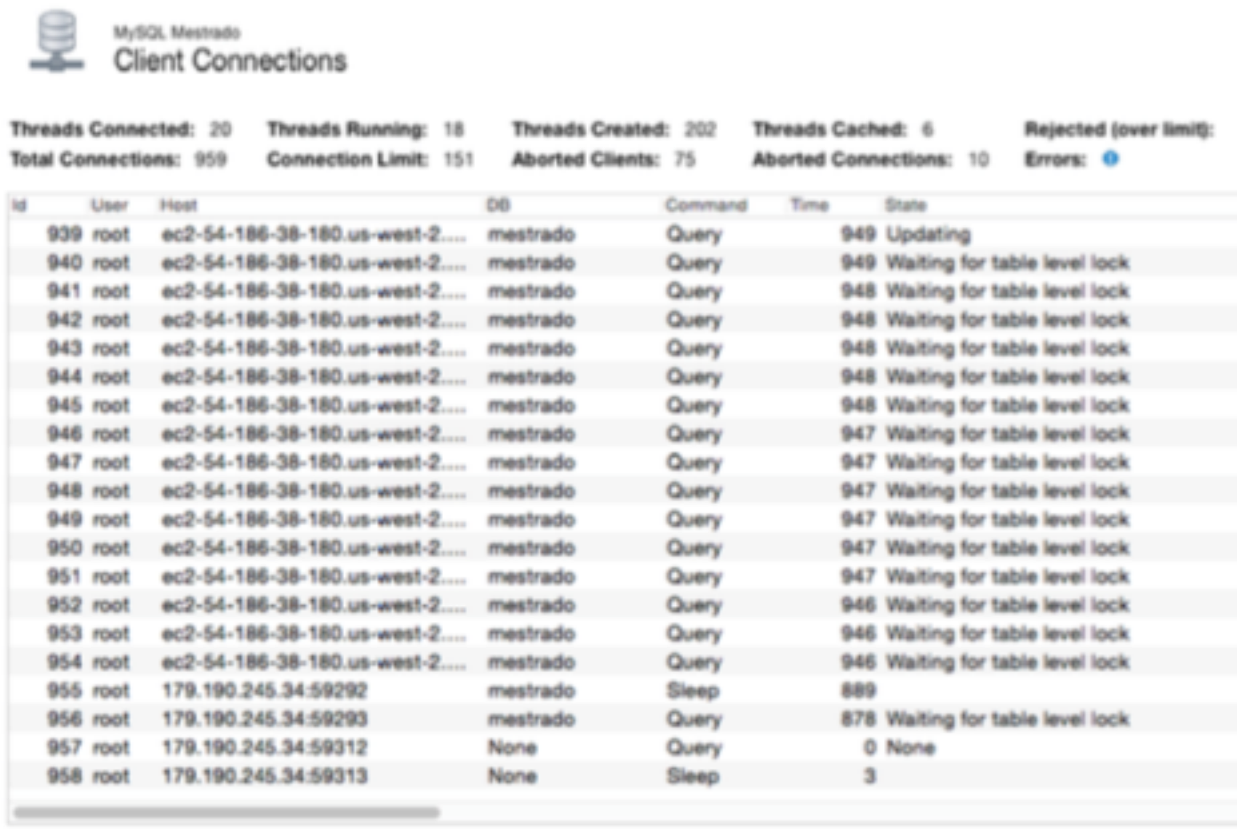


Figure 22: Edit Tags - Dataset Sizes vs Time (ms).



MySQL Mestrado
Client Connections

Threads Connected: 20 Threads Running: 18 Threads Created: 202 Threads Cached: 6 Rejected (over limit):
Total Connections: 959 Connection Limit: 151 Aborted Clients: 75 Aborted Connections: 10 Errors:

Id	User	Host	DB	Command	Time	State
939	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	949	Updating
940	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	949	Waiting for table level lock
941	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	948	Waiting for table level lock
942	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	948	Waiting for table level lock
943	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	948	Waiting for table level lock
944	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	948	Waiting for table level lock
945	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	948	Waiting for table level lock
946	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	947	Waiting for table level lock
947	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	947	Waiting for table level lock
948	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	947	Waiting for table level lock
949	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	947	Waiting for table level lock
950	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	947	Waiting for table level lock
951	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	947	Waiting for table level lock
952	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	946	Waiting for table level lock
953	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	946	Waiting for table level lock
954	root	ec2-54-186-38-180.us-west-2....	mestrado	Query	946	Waiting for table level lock
955	root	179.190.245.34:59292	mestrado	Sleep	889	
956	root	179.190.245.34:59293	mestrado	Query	878	Waiting for table level lock
957	root	179.190.245.34:59312	None	Query	0	None
958	root	179.190.245.34:59313	None	Sleep	3	

Figure 23: MySLQ Job Queueing.

5. VERIFY SLA VIOLATION

- Operation 03:
- SLA is broken on 300K and 3KK datasets

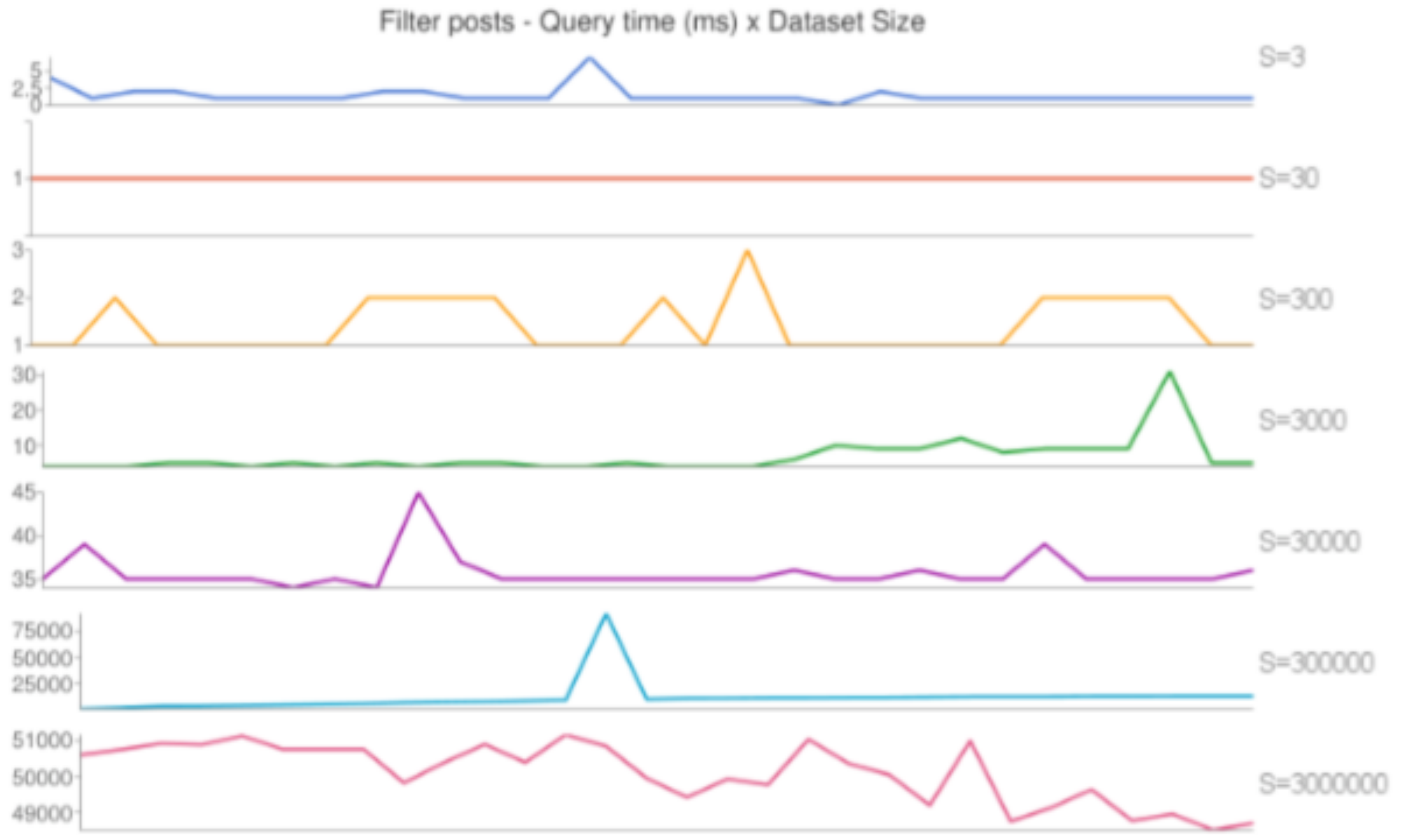


Figure 24: Filter captured posts by filters - Dataset Sizes vs Time (ms).

6. PROPOSE ARCHITECTURAL CHANGES - OPERATION 02

- Operation 02: Improving DB architecture works?
  - 1. Switch table Engine: InnoDB vs MyISAM - no results
  - 2. Build FTS index on tags column - no results
  - 3. Change the way tags are stored - improvements, but SLA remains broken
  - 4. FTS Indexes + changes: now "ok".

- MyISAM
  - No foreign keys and cascading deletes/updates
  - No transactional integrity (ACID compliance)
  - No rollback abilities
  - Row limit of 4,284,867,296 rows
  - Maximum of 64 indexes per row

6. PROPOSE ARCHITECTURAL CHANGES – OPERATION 02

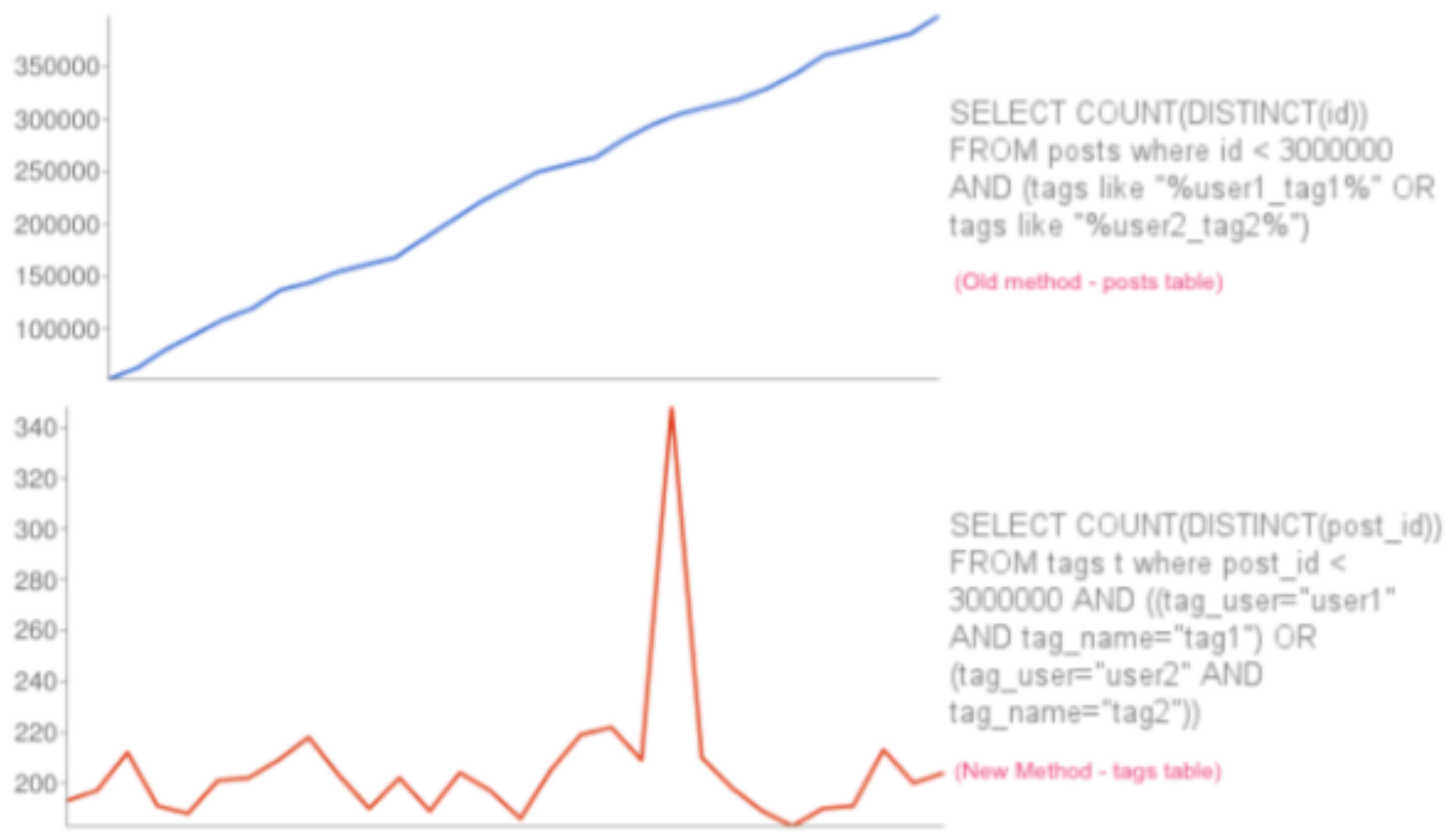


Figure 27: Retrieve posts to be modified - (on Tags table / MyISAM) vs Retrieve posts to be modified (on posts table / InnoDB) - 3 million posts dataset.

- SLA still "cant be met". It enables to search, what about *writing* the changes?
  - Every operation is limited by the number of ops in disk that the database is able to execute.
  - SLA Renegotiation is possible? For instance, users may agree that 10.000 tags/second is a satisfiable level

6. PROPOSE ARCHITECTURAL CHANGES – OPERATION 03

- Operation 03: Improving DB architecture works?
  - 1. Switch table Engine: InnoDB vs MyISAM - no results;
  - 2. Build FTS index on message column;
  - 3. Build BTrees indexes on Sentiment and other searchable fields;
  - SLA is not broken anymore, but results are wrong!

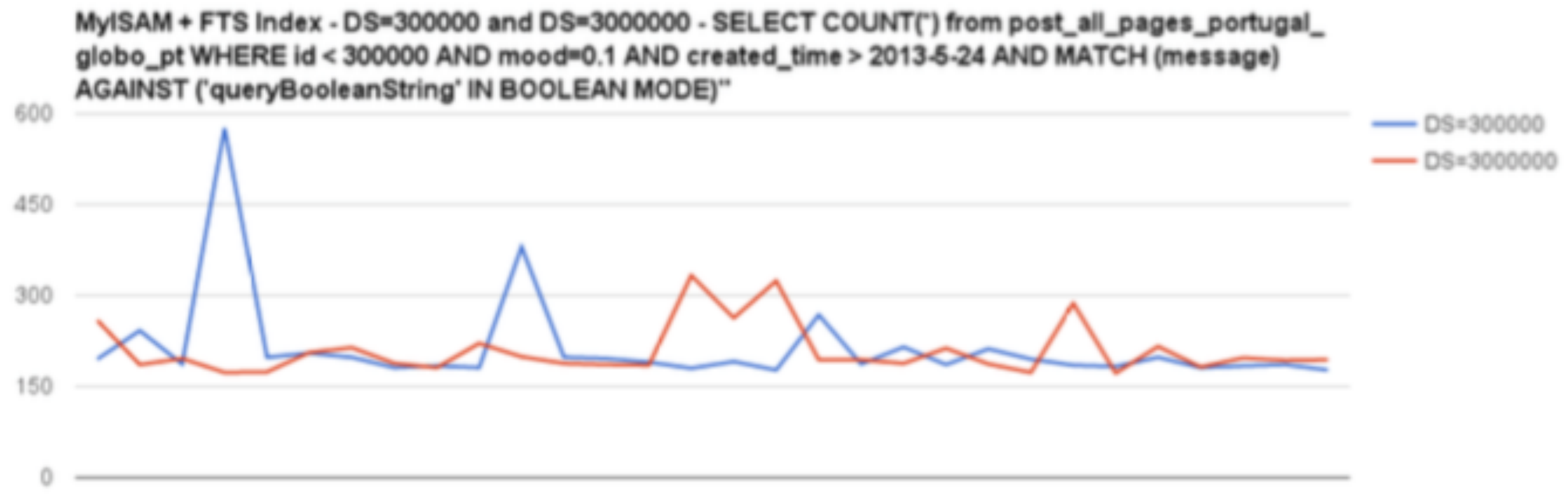


Figure 28: Searching on posts table - New Format: MyISAM and FTS Index.

6. PROPOSE ARCHITECTURAL CHANGES – OPERATION 03

- Searching = Stemming?
- Like Operator:
  - “Lay” => Play, laying, Lay, Lays. Play is not desired at this context.
- Match Against operator
  - “Lay” => Lay. What about “Laying" or Lays or Laying?
- Both are not what the user expects.

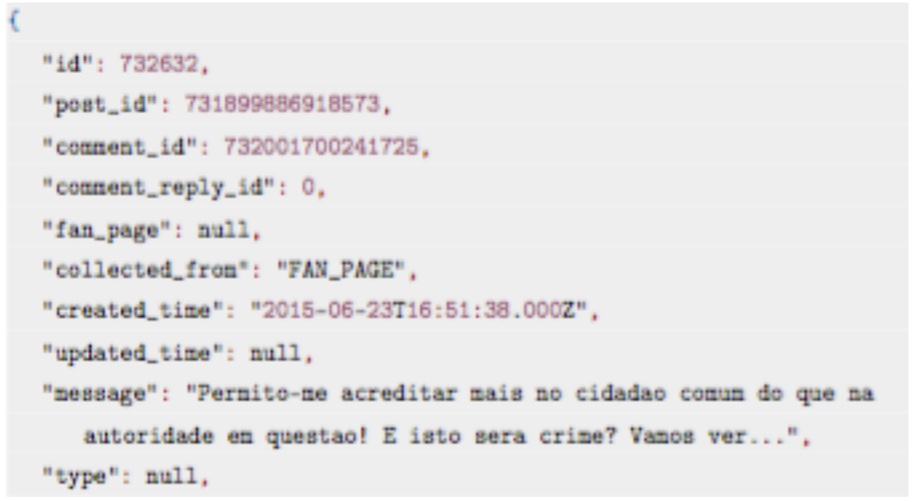
6. PROPOSE ARCHITECTURAL CHANGES - OPERATION 03

- Solution: Depend on a external plugins to do stemming
    - Develop your own solutions, reinvent the wheel.
    - No community maintenance.

- Apache Solr, Lucene, Amazon Cloudsearch and Elasticsearch are **Search Engines** that provide **fulltext-search and Stemming** as main features.
- After an extensive literature review, discussing in popular technical forums and analyzing benchmarks (STACKOVERFLOWELASTIC, 2015) (SOLRVSELASTICSEARCH, 2015) (QUORAELASTIC, 2015), Elasticsearch (a.k.a. Elastic) seemed to be a good alternative to the problem that the application struggled with MySQL.

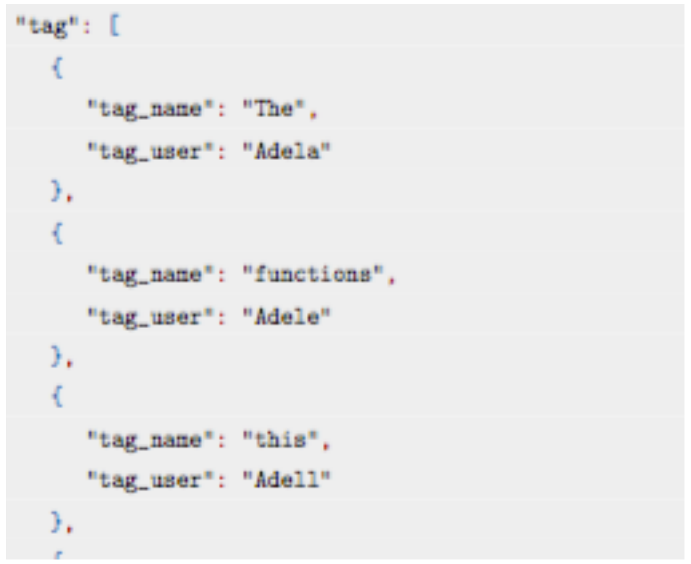
7. PROPOSE A NEW DB ARCHITECTURE

- New Data Model

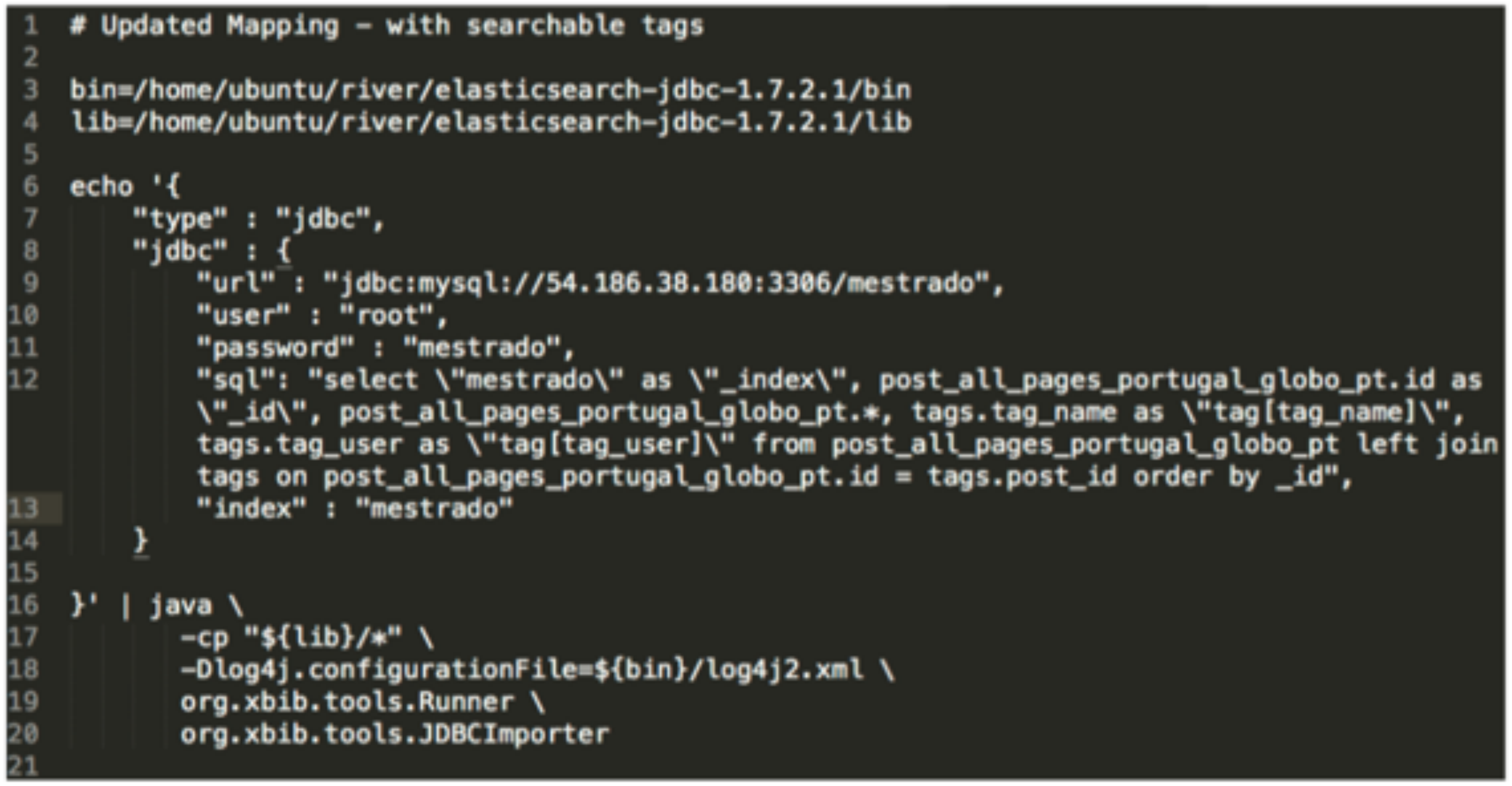


```
{
  "id": 732632,
  "post_id": 731899886918573,
  "comment_id": 732001700241725,
  "comment_reply_id": 0,
  "fan_page": null,
  "collected_from": "FAN_PAGE",
  "created_time": "2015-06-23T16:51:38.000Z",
  "updated_time": null,
  "message": "Permito-me acreditar mais no cidadao comum do que na
     autoridade em questao! E isto sera crime? Vamos ver...",
  "type": null,
```

• • •



```
"tag": [
   {
     "tag_name": "The",
     "tag_user": "Adela"
   },
   {
     "tag_name": "functions",
     "tag_user": "Adele"
   },
   {
     "tag_name": "this",
     "tag_user": "Adell"
   },
```



```
# Updated Mapping - with searchable tags

bin=/home/ubuntu/river/elasticsearch-jdbc-1.7.2.1/bin
lib=/home/ubuntu/river/elasticsearch-jdbc-1.7.2.1/lib

echo '{
    "type" : "jdbc",
    "jdbc" : {
        "url" : "jdbc:mysql://54.186.38.180:3306/mestrado",
        "user" : "root",
        "password" : "mestrado",
        "sql": "select \"mestrado\" as \"_index\", post_all_pages_portugal_globo_pt.id as
        \"_id\", post_all_pages_portugal_globo_pt.*, tags.tag_name as \"tag[tag_name]\",
        tags.tag_user as \"tag[tag_user]\" from post_all_pages_portugal_globo_pt left join
        tags on post_all_pages_portugal_globo_pt.id = tags.post_id order by _id",
        "index" : "mestrado"
    }

}' | java \
       -cp "${lib}/*" \
       -Dlog4j.configurationFile=${bin}/log4j2.xml \
       org.xbib.tools.Runner \
       org.xbib.tools.JDBCImporter
```

Figure 29: Mapping - MySQL to ES.

https://github.com/jprante/elasticsearch-jdbc

- + PT-BR Stemming: https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-lang-analyzer.html

8. EXECUTE OPERATIONS FROM A HISTORICAL POINT ON

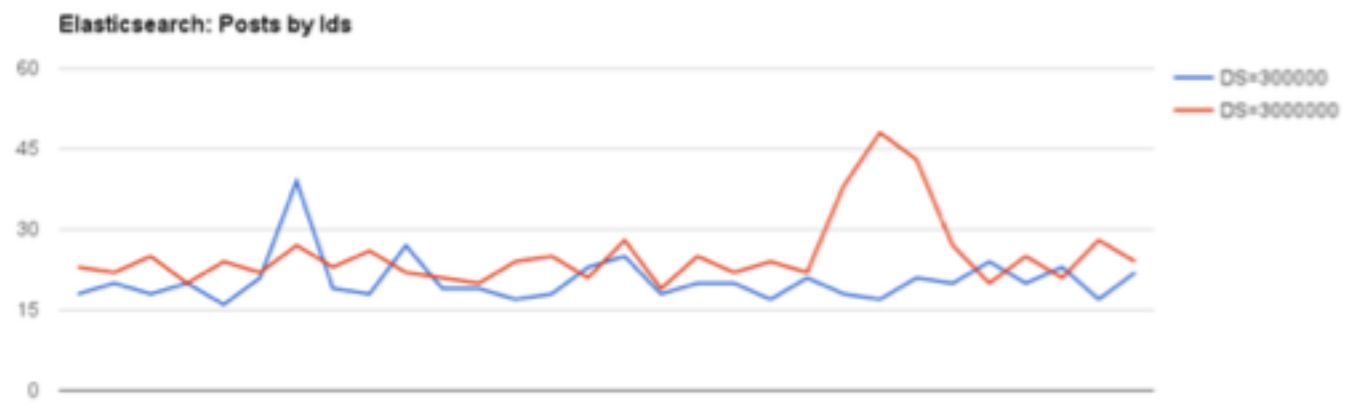


Figure 30: Elasticsearch architecture: Retrieve posts by Id.

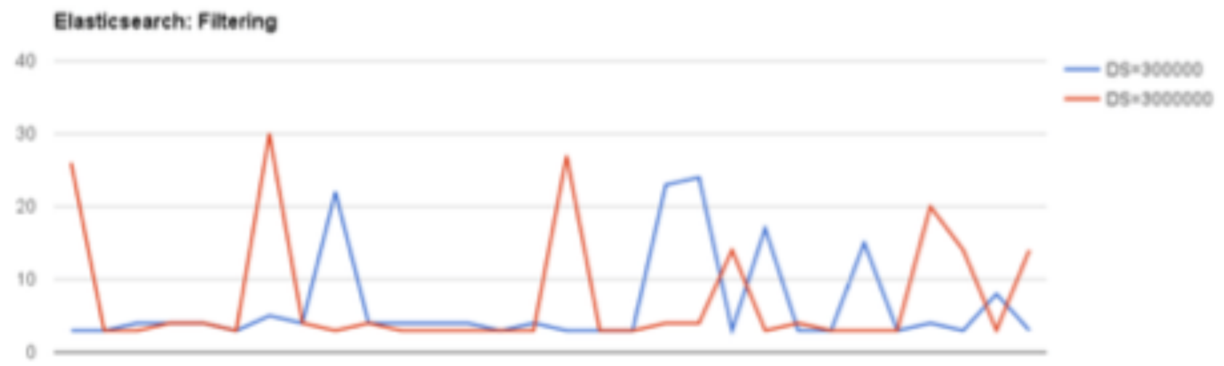


Figure 32: Elasticsearch architecture: Filter posts.

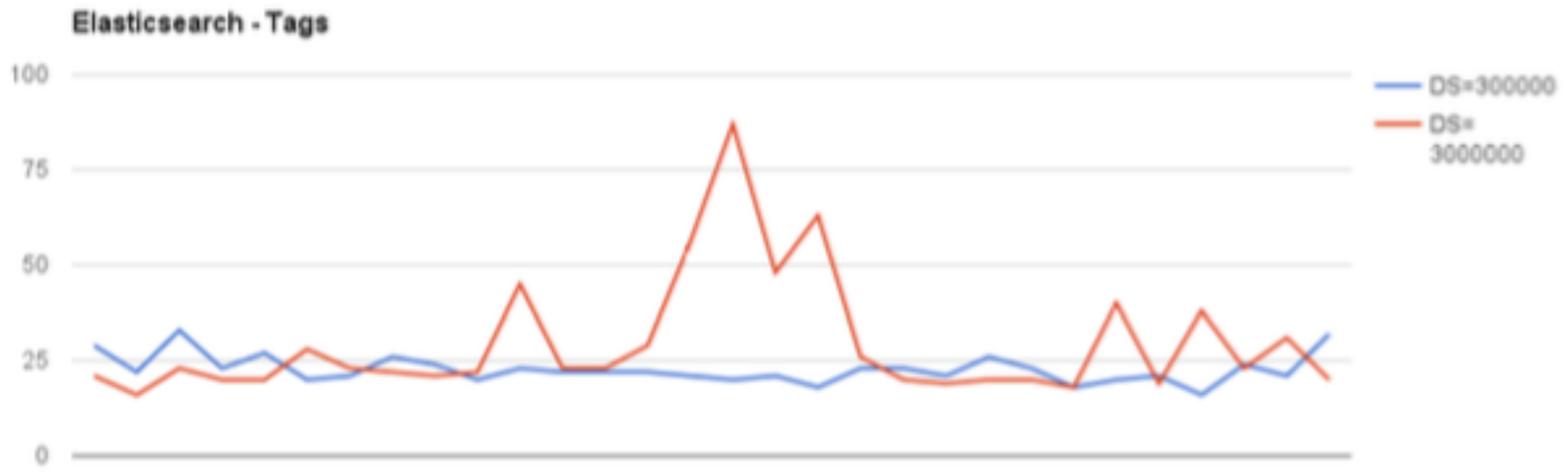


Figure 31: Elasticsearch architecture: Update by tags.

WHAT ELSE IT ENABLES?

- Build word clouds (code level or build temporary tables with MySQL) - Bucketing, in-memory-processing, aggregations and term-vectors
- FTS Support in several languages
- "Did you mean?" feature with term suggester for Elasticsearch
- Deep paging for the proposed application architecture

CONCLUSIONS - DATABASE TRANSITIONS

- Not always it is necessary to completely switch the database
- Using a NoSQL architecture (as Elasticsearch in the context of this app) can help to leverage user experience, QoS and accelerate application development.
- Database transitions should be test-oriented to assure that it will not change the way the application works.
- Verify if it's able to handle the real workload - handle production requests in parallel for a while.

CONCLUSIONS - CONTRIBUTIONS

- A Systematic Mapping study about database transitions published on an interna-tional conference. (LEAL; MUSICANTE, 2015)(Qualis/CC:B1)
- Development of manpower able to work with database transition scenarios from relational databases to NoSQL environments. As NoSQL is a relatively recent con- cept, not many professionals are experienced with database transitioning scenarios to these technologies;
- A set of guidelines to guide and assess database transitions from relational to NoSQL databases;
- The core architecture, based on polyglot persistence approach, of a social-media monitoring application;
- A set of Java-implemented load test scenarios that can be adapted to assess the QoS of applications.