**Marco Bassi**

Lorrainestrasse 23

3013 Bern

Switzerland

**Data Science Project**

**Assessment of Database Performance Degradation**

**Conceptual Design Report**

**22nd September 2019**

# ABSTRACT

Today’s commercial data management systems (short: database) often operate in complex application environments, with the databases generally being the pivotal component. Thus, if the database performance degrades, then the system’s overall performance is at stake. Tight monitoring of the database’s performance is mandatory in such an environment. To facilitate database monitoring, every commercial database system is equipped with an impressive number of system statistics, allowing to look at every aspect of the database’s operation.

For the data science project I will describe on the following pages, I use database statistics to assess a suspected performance degradation, which occurred after an upgrade of the database infrastructure on the hosting server. After the upgrade we noticed deteriorations of the response times of several application components. In order to held responsible the database manufacturer and the provider of the hosting server, we have to provide evidence that the deteriorations indeed originate on the database.

**Deliverables M1:** Conceptual Design Report, (this document), GitHub Repository including also Jupyter Notebook and poster from Module 2.

**Expected effort:** about 30 hours,

# 

# 

# TABLE OF CONTENTS

ABSTRACT 1

TABLE OF CONTENTS 2

OBJECTIVE 3

METHODS 4

DATA 4

METADATA 1

DATA QUALITY 1

DATA FLOW 1

From source to final plot 1

DATA MODELS 1

RISKS 2

PRELIMINARY STUDIES 2

CONCLUSIONS 2

REFERENCES 2

Appendix 2

# OBJECTIVE

The database examined is the integration database of the Swiss Federal Railways (SBB) CUS platform. CUS is the acronym for Customer Service. It is a datahub for real-time data of swiss public transport.

The data is provided by the participating transport companies, a few tens, from Switzerland and the neighboring countries. It is enriched with data taken from internal and external information system. The compiled data is made available to our partners, and also to various traveler information systems; e.g. screen displays in stations and in vehicles; announcements by loudspeakers; internet websites and applications, etc.

The integration and the production database each run on their own, dedicated platform; an Oracle Database Appliance (ODA). And each database consists of two instances which run in cluster mode (Oracle Real Application Cluster RAC). The integration environment’s main purpose is to enable tests of a new release before it is deployed onto the production platform. A new release must be tested for functional and technical correctness, and for its performance under high load.

After an upgrade in spring 2019 of the grid infrastructure of the integration database’s hosting server, this database suffered from serious performance problems. The performance issues mostly manifested themselves as significantly increased response times of several application components on the one hand. And by massively increased waits related to the cluster operation of the databases, on the other hand. With this degradation the release tests were at risk.

In beginning of August 2019, a number of patches were applied to the hosting server, as a corrective action. The analysis' goal is to test if the patching resolved the performance degradation. Or if it prevails, in which case the release tests would be compromised. As I cannot use gathered system statistics from pre-upgrade days, I will compare selected performance statistics of the integration database to the production database.

# METHODS

The analysis was executed in an Anaconda environment:

* Anaconda Navigator 1.9.7
* JupytherLab 1.0.2
* Conda Packages
  + r 3.6.0 (R 3.6.1)
  + r-irkernel 0.8.15
  + r-data.table 1.12.2
  + r-broom 0.5.2
  + stats 3.6.1 (r-essentials 3.6.0)
  + r-ggplot2 3.1.1
  + r-reshape2 1.4.3

The computations have been performed on my Lenovo P50 notebook

* Intel Core i7-6820HQ (x64-based)
* 32 GB RAM
* Windows 10 Enterprise, Version 1803

As I have unpaired samples and cannot presume any distribution of the data, I have to rely on non-parametric tests. I choose the Wilcoxon Rank Sum Test (one-tailed and two-sided).

# DATA

Oracle measures hundreds of statistics in real time, and makes them available by so-called *dynamic performance views*. E.g. statistics related to the database system can be found in the view SYS.SYSSTAT. The values in these performance views are running sums. Upon restart of a database instance, all its statistics are reset to zero.

On every hour a snapshot of all dynamic performance views is made and stored in the *static performance views*. For SYS.SYSSAT the corresponding static view is SYS.DBA\_HIST\_SYSSTAT; see figure 1 DBA\_HIST\_SYSSTAT. Of this view I need the following columns:

snap\_id: identifies a snapshot interval. Snapshots of all database instances, taken at the same hour, have identical snap\_id

instance\_number: id of the database instance for which a statistic was measured

stat\_name: name of the statistics

value: value of the running sum of the statistic at the end of the snapshot interval.

To get the snapshot details, e.g. begin and end of the snapshot intervals, it must be joined with the DBA\_HIST\_SNAPSHOT view. This view provides all the snapshot details. I will just need the columns

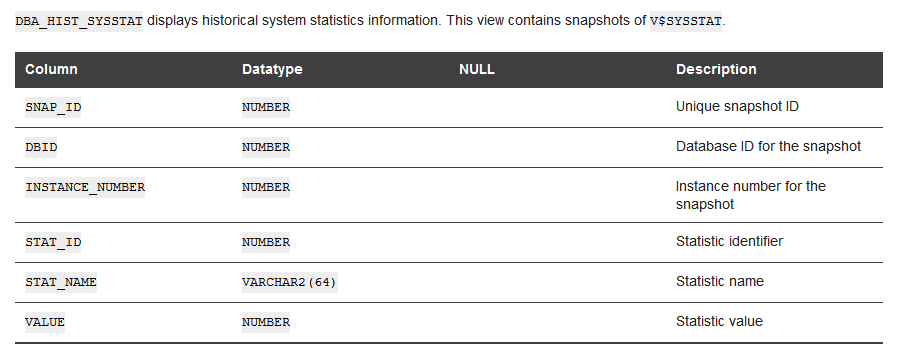
1 DBA\_HIST\_SYSSTAT

begin\_interval\_time: timestamp of the beginning of the snapshot interval

end\_interval\_time: timestamp of the end of the snapshot interval

The statistics data was selected using the SQL statement below:

**alter session set nls\_timestamp\_format = 'YYYY-MM-DD HH24:MI:SS';**

**select begin\_interval\_time, end\_interval\_time, snap\_id,   
instance\_number, stat\_name, value  
from dba\_hist\_sysstat  
natural join dba\_hist\_snapshot  
where begin\_interval\_time between  
 timestamp '2019-08-22 00:00:00'  
 and timestamp '2019-08-29 08:00:00'   
order by begin\_interval\_time, instance\_number, stat\_name;**

I have choosen the time range such that both the integration and production database have application version 5.11.1 1.180.1 (on August 29, 2019 at 09:00 CEST, application version 5.12.0 1.188.1 was installed on the integration environment).

The data was exported to semicolon-separated text files:

* dba\_hist\_sysstat.inte.dsv for the integration database, and
* dba\_hist\_sysstat.prod.dsv for the production database.

The files are stored in the subdirectory Data-Science-Project\project.1\statistiken.2019-08-22T0000-bis-2019-08-28T1000\data of the GitHub repository mbassi1364/CAS-Applied-Data-Science. For the URL see section REFERENCES at the end of the document.

Extract from the integration statistics:



As there are hundreds of statistics, I limited the analysis to a small subset, chosen such that I can address some conclusive aspects of database performance.

Wait Classes

* application wait time
* cluster wait time
* concurrency wait time
* user I/O wait time

Database Load

* db block changes
* enqueue requests
* execute count
* global enqueue gets async
* global enqueue gets sync
* parse count (total)
* user calls

Global Cache Activity

* gc cr blocks received
* gc current blocks received
* gc local grants
* gc read waits
* gc remote grants
* gcs messages sent

Global Cache Wait Events

* gc cr block flush time
* gc cr block receive time
* gc current block flush time
* gc current block receive time
* gc current block send time
* gc read wait time
* global enqueue get time

# METADATA

1. Both database platforms are Oracle Data Appliances X5-2-HA, with Oracle RDBMS Enterprise Edition 11.2.0.4.0 for Linux 64 bit, and Oracle Real Application Cluster.
2. On both databases the same application version 5.11.1 1.180.1 is installed.
3. None of the integration and production database instances was restarted in the period to be analysed. This precondition is not mandatory, but makes analysis more straightforward.
4. The statistics are described in the Oracle document Oracle Database Reference, 11g Release 2 (11.2), August 2015; section Statistics Descriptions. See References.
5. The Oracle dynamic performance views DBA\_HIST\_SYSSTAT (historicized system statistics) and DBA\_HIST\_SNAPSHOT (historicized snapshot intervals) are described in the same document, section Static Data Dictionary Views.

The metadata is described in the markdown document readme.md, subfolder Data-Science-Project\project.1\statistiken.2019-08-22T0000-bis-2019-08-28T1000\about.data of the Github repository CAS-Applied-Data-Science. See section REFERENCES at the end of the document.

# DATA QUALITY

What are the quality requirements you have to reach your analysis goal (precision ...)?

Are they met? If not, do you expect a significant impact on your results,

Any measures to improve the quality?

# DATA FLOW

# From source to final plot

# DATA MODELS

Conceptual

Logical (with dataframes and with databases)

Physical (infrastructure needs)

# RISKS

What can go wrong?

When this and that goes wrong, what measures do you have?

What will be the impact?

# PRELIMINARY STUDIES

From module 2

# CONCLUSIONS

….

# REFERENCES

1 Oracle® Database Reference 11g Release 2 (11.2), E40402-18, August 2015  
https://docs.oracle.com/cd/E11882\_01/server.112/e40402/toc.htm

2 GitHub Repository mbassi1364/CAS-Applied-Data-Science  
https://github.com/mbassi1364/CAS-Applied-Data-Science

3

....

# Appendix

Swiss National Science Foundation data management plan