Marco Bassi
Lorrainestrasse 23
3013 Bern
Switzerland

Data Science Project

Assessment of Database Performance Degradation

Conceptual Design Report

24th September 2019

Version 1.1

ABSTRACT

Today's commercial data management systems (short: database) often operate in complex application environments, with the database generally being the pivotal component. Thus, if the database performance degrades, then the system's overall performance is at stake. Tight performance monitoring 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 use database statistics to assess a suspected database performance degradation, which occurred after an upgrade of the database's 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.

TABLE OF CONTENTS

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 homologized and enriched with data taken from internal and external information systems. The compiled data is made available to our partners, and 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). Both databases consist of two database instances, running in cluster mode (Oracle Real Application Cluster RAC). The integration environment's main purpose is for tests of a new releases before they are deployed onto the production environment. Every 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 server hosting the integration database, 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 database, on the other hand. With this degradation the release tests were at risk.

In beginning of August 2019, 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 still prevails, in which case the release tests would be compromised. As I cannot use gathered system statistics from pre-upgrade days, I will compare the integration database to the production database.

METHODS

Environment

The analysis is 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 are 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 will rely on non-parametric tests. I have chosen the Wilcoxon Rank Sum Test (one-tailed and two-sided).

Procedure

Compare Waiting Classes: Every database event causing a session to wait, is assigned to a waiting class. A waiting class aggregates all waiting times of events assigned to it. Waiting classes provide a high-level view of database activities, showing were sessions spend and loose time. Comparison is done between waiting classes within one database on the one hand, and between the databases on the other hand. Waiting classes will not be compared between database instances.
The main waiting classes are application, concurrency, cluster and user I/O.

Compare Database Load: Some system statistics are indicators for database load, e.g. the number of data blocks changed, or the number of calls that executed a SQL statement. The statistics are compared between the databases.

Compare Cluster Statistics: Cluster statistics are related to operations due to the RAC setup of the databases. Both instances of each database share the data. When one database instance requests a data block, and this data block is currently held in the local cache of the other database instance, a complex protocol is followed. The main concept behind this protocol is the so-called global cache.

I compare statistics measuring the global cache load, and statistics measuring the waiting times spent during global cache-related activities. Comparison is done between the databases.

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.GV$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 created and stored in the *static performance views*. For SYS.GV$SYSSAT the corresponding static view is SYS.DBA_HIST_SYSSTAT; see figure 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: statistic name

value: value of the running sum of the statistic, at the end of the snapshot interval, i.e. at the moment the snapshot was created.

To get the snapshot details, DBA_HIST_SYSSTAT must be joined with the DBA_HIST_SNAPSHOT view. The columns I need are:

begin_interval_time: timestamp at the beginning of the snapshot interval

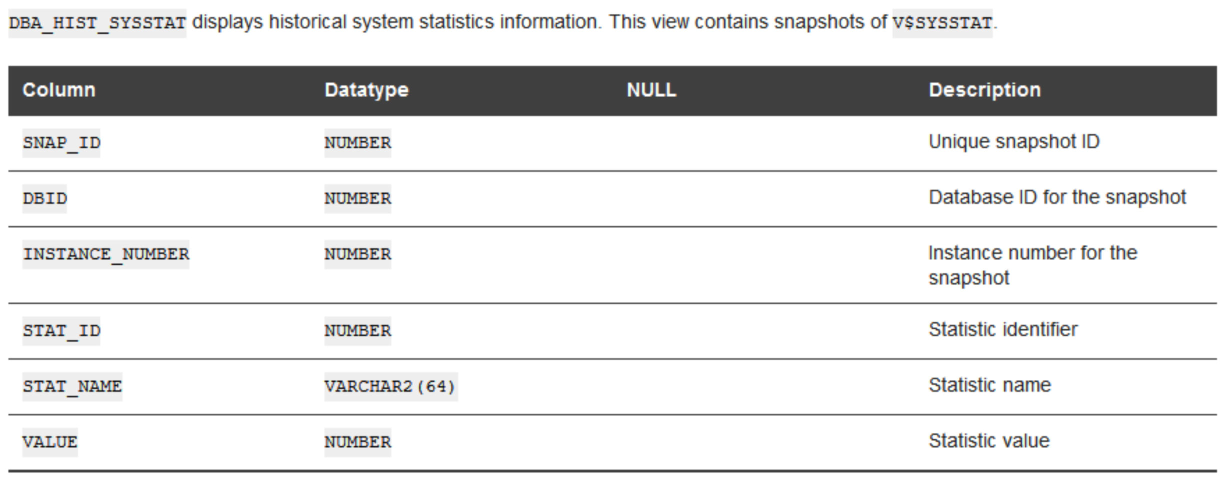
end_interval_time: timestamp at the end of the snapshot interval

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 chosen 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).


DBA_HIST_SYSSTAT displays historical system statistics information. This view contains snapshots of V$SYSSTAT.

Column	Datatype	NULL	Description
SNAP_ID	NUMBER		Unique snapshot ID
DBID	NUMBER		Database ID for the snapshot
INSTANCE_NUMBER	NUMBER		Instance number for the snapshot
STAT_ID	NUMBER		Statistic identifier
STAT_NAME	VARCHAR2(64)		Statistic name
VALUE	NUMBER		Statistic value

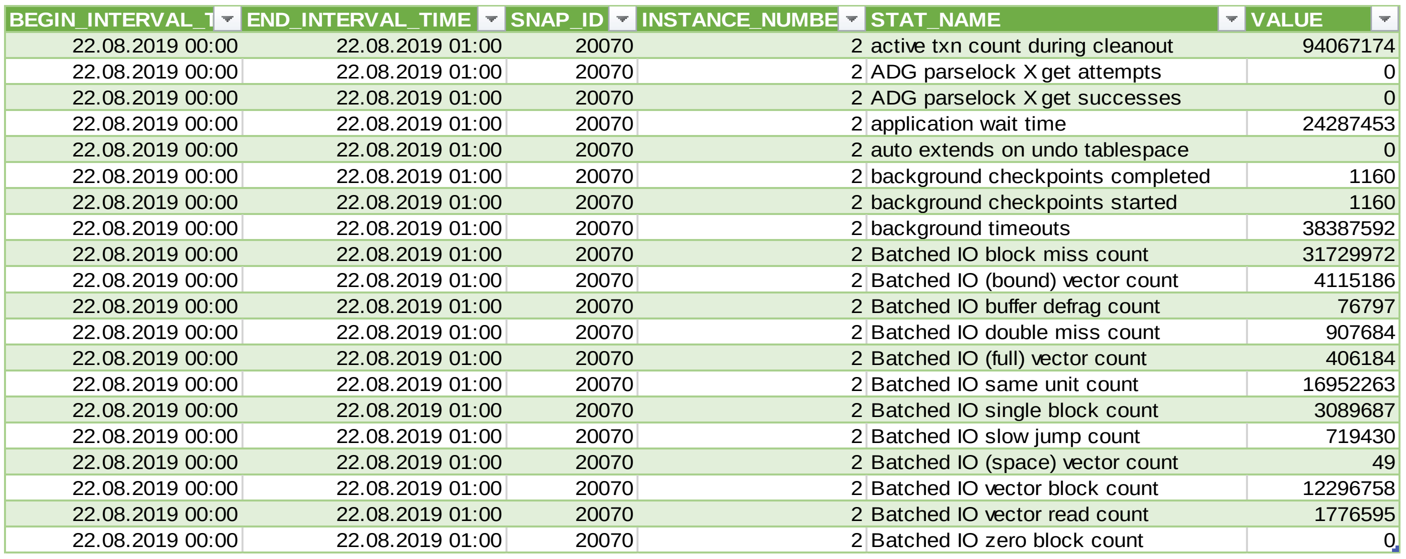
Figure 1 DBA_HIST_SYSSTAT

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:



BEGIN_INTERVAL_T	END_INTERVAL_TIME	SNAP_ID	INSTANCE_NUMBE	STAT_NAME	VALUE
22.08.2019 00:00	22.08.2019 01:00	20070	2	active txn count during cleanout	94067174
22.08.2019 00:00	22.08.2019 01:00	20070	2	ADG parselock X get attempts	0
22.08.2019 00:00	22.08.2019 01:00	20070	2	ADG parselock X get successes	0
22.08.2019 00:00	22.08.2019 01:00	20070	2	application wait time	24287453
22.08.2019 00:00	22.08.2019 01:00	20070	2	auto extends on undo tablespace	0
22.08.2019 00:00	22.08.2019 01:00	20070	2	background checkpoints completed	1160
22.08.2019 00:00	22.08.2019 01:00	20070	2	background checkpoints started	1160
22.08.2019 00:00	22.08.2019 01:00	20070	2	background timeouts	38387592
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO block miss count	31729972
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO (bound) vector count	4115186
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO buffer defrag count	76797
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO double miss count	907684
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO (full) vector count	406184
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO same unit count	16952263
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO single block count	3089687
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO slow jump count	719430
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO (space) vector count	49
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO vector block count	12296758
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO vector read count	1776595
22.08.2019 00:00	22.08.2019 01:00	20070	2	Batched IO zero block count	0

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

Waiting 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 the 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.
6. The character set encoding scheme of the data export files is UTF-8.

The metadata is described in the markdown document metadata.readme.md, stored in the 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

Time Fuzziness

The real-time statistics are collected in the database's dynamic memory, for each database instance separately. The dynamic performance views, therefore, are not views defined on a database table, but only a convenient way to look at the data. This approach is fast; however, there is a major drawback: There is nothing like a *System Change Number* enabling a consistent view of the data. Therefore, if you compare statistics taken from two or more dynamic performance views, or the same statistic for two or more database instances, you have no guarantee that the data represents the database's or database instance's state at one moment in time. Real-time performance monitoring always must allow for a certain amount of time fuzziness. In times of high load, however, the time fuzziness may increase to a significant degree, which unfortunately cannot be assessed in a straightforward way.

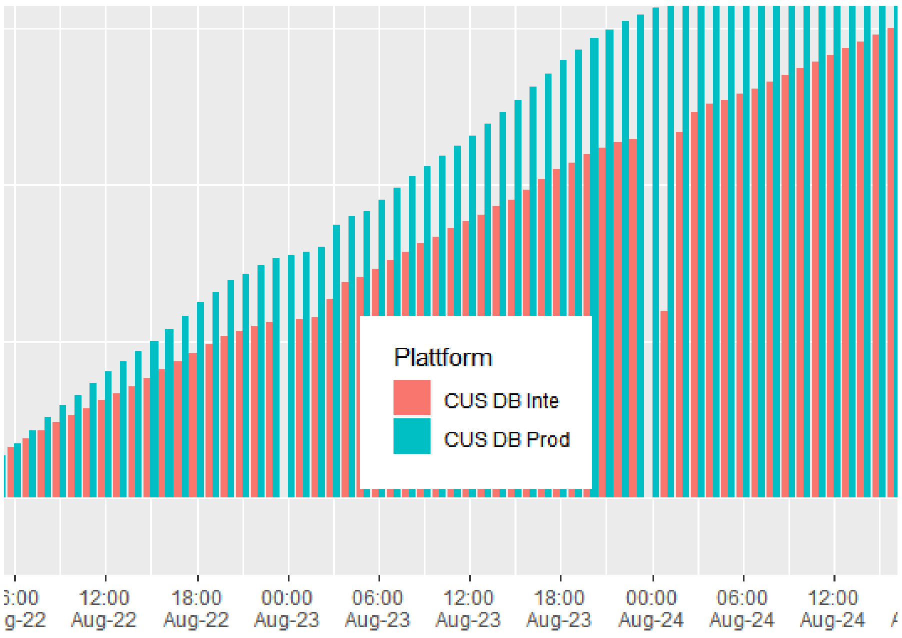


Figure 2 Snapshot Creation

Snapshot Creation

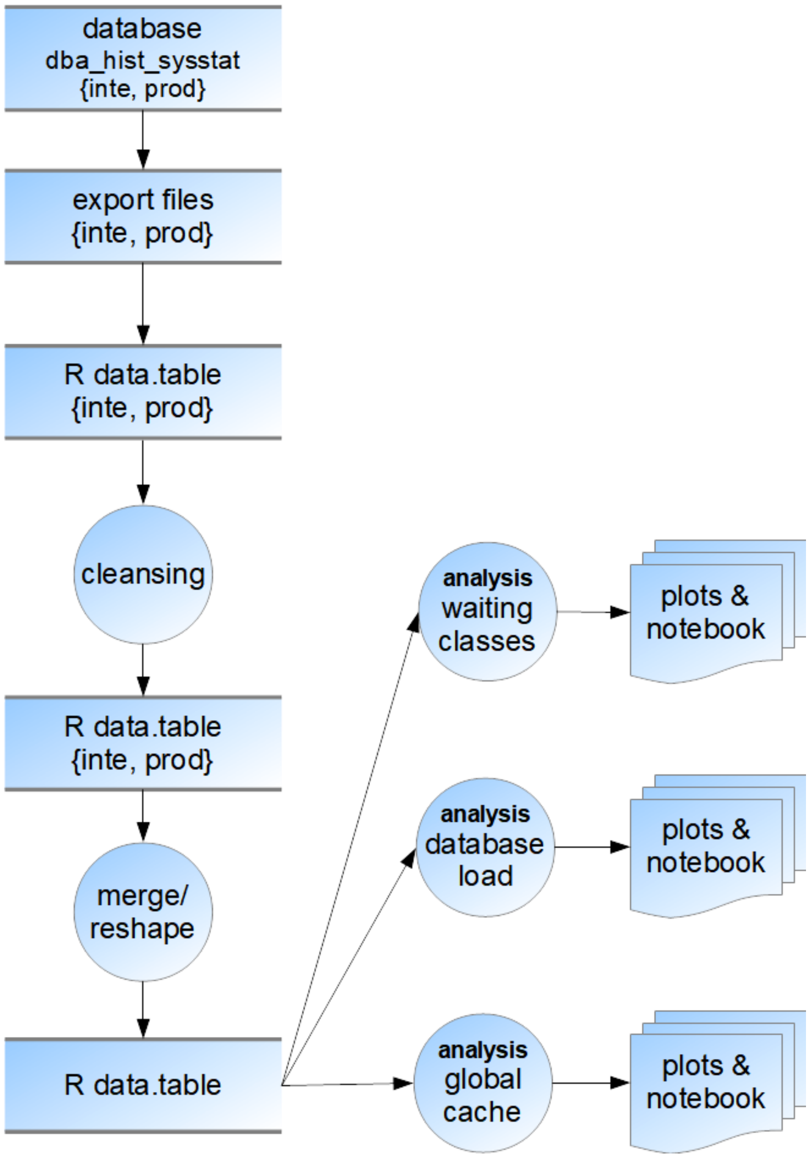
The problem of time fuzziness is mitigated when you work with hourly snapshots. But there is a new problem: The snapshots are created by a database background process. This is a low-priority process. Therefore, in times of high database load, creating snapshots may be delayed by several minutes. In critical database situations lasting an hour or more, statistics data will be lost; see figure Figure 2 Snapshot Creation.

As a consequence, snapshot intervals must be examined closely before starting an analysis. There are several approaches:

1. Check begin, end, and duration of snapshot intervals (database view DBA_HIST_SNAPSHOT).
2. Plot a time series of selected statistics.
3. Check flush_elapsed and error_count (database view DBA_HIST_SNAPSHOT).

As snapshots are created for each database instance separately, snapshot interval checking must be performed separately for each database instance.

DATA FLOW



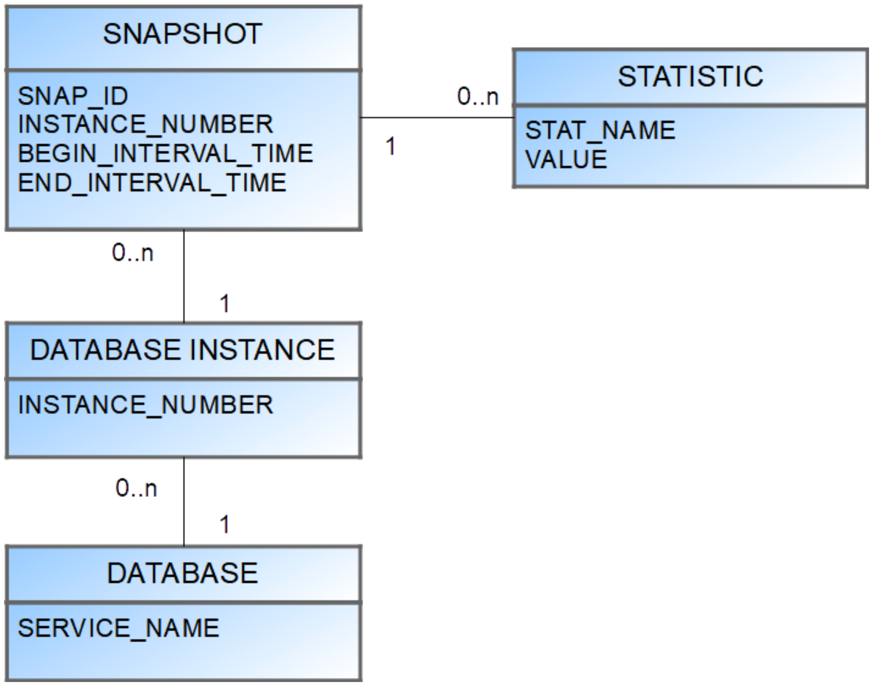
Procedure Followed for the Data Analysis

1. Export statistics from databases into CSV files; separate files for integration and production data
2. Import CSV files into R data.tables; separate data.tables for integration and production data
3. Data cleansing into new R data.tables; separate cleansing for integration and production data
4. Merge R data.tables for integration and production data into one R data.table, and reshape the data into long format.

5. Perform separate data analysis for
    a. waiting time classes
    b. database load statistics
    c. global cache statistics

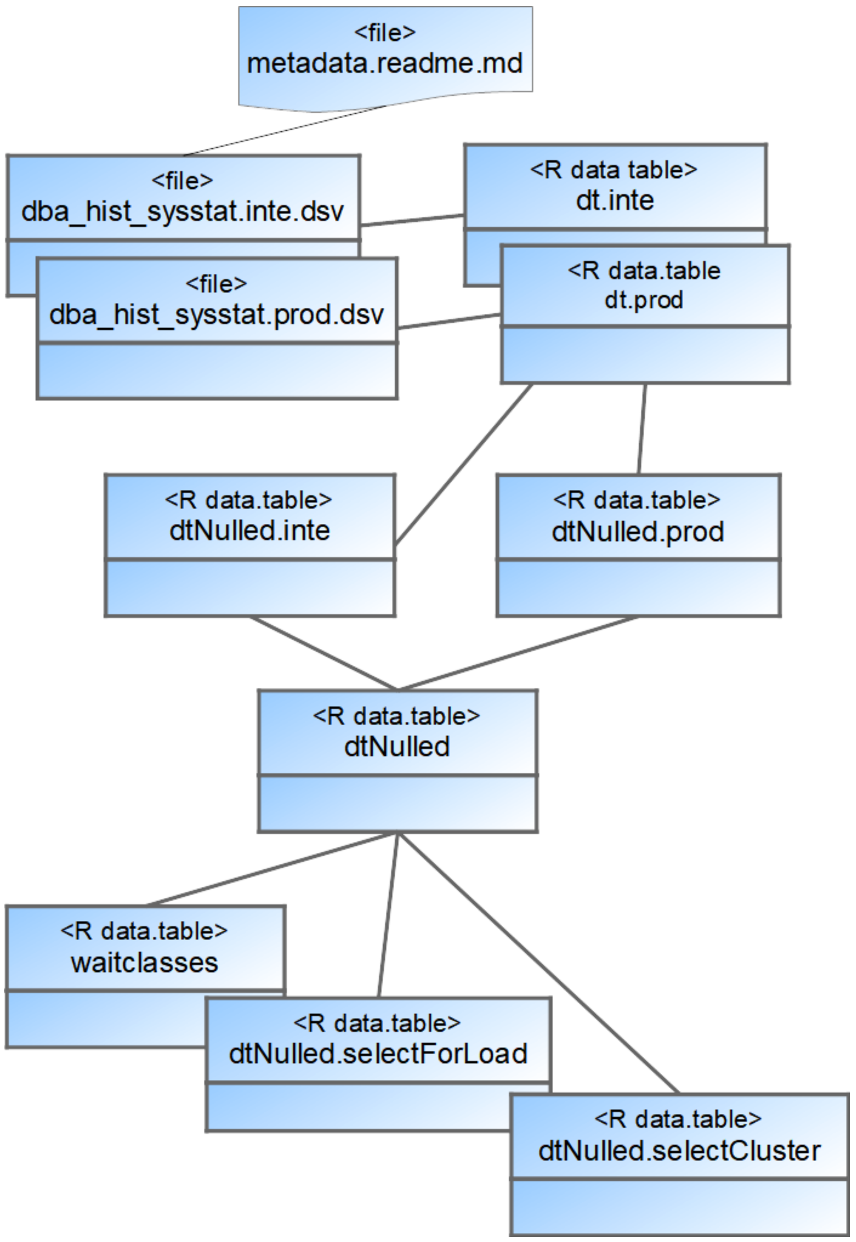
DATA MODELS

Conceptual Data Model



The conceptual mode is fairly simple. The central concept is the statistic, having a name and a value. A statistic instantiation is a concrete measurement. It must be assigned a snapshot instantiation, and so on …

Logical Data Model



The data is read from the data files (one for the integration database, and one for the production database) into R data.table objects, one for each database. Data cleansing results in new data.table objects, still one per database. They are then merged into one data.table object, and reshaped into long format.

For the various analyses, a dedicated data.table is created by subsetting the merged one.

The metadata is stored in an unstructured text file.

Physical Data Model

Export to File

- Export from integration database: 232897 Records
- Export file integration database: dba_hist_sysstat.inte.dsv (18.3 MB)
- Export from production database: 239008 Records
- Export file production database: dba_hist_sysstat.prod.dsv (18.8 MB)

Data exported using Oracle SQL Developer 18.1.0 for Microsoft Windows 10 (x64)

Export Files

- Format: Text file
- Field separator: ";" (semicolon)
- 1st line: field names
- 2nd line and below: records
- Field 1: BEGIN_INTERVAL_TIME; timestamp YYYY-MM-DD HH24:MI:SS
- Field 2: END_INTERVAL_TIME; timestamp YYYY-MM-DD HH24:MI:SS
- Field 3: SNAP_ID; snapshot ID, non-negative integer
- Field 4: INSTANCE_NUMBER; ID of the database instance, non-negative integer
- Field 5: STAT_NAME; statistics' name
- Field 6: VALUE; statistics' value, integer

R Data Tables

- Import from export file production
```
239008 obs. of  6 variables:
BEGIN_INTERVAL_TIME: chr  "2019-08-22 00:00:14" "2019-08-22 00:00:14" ...
END_INTERVAL_TIME  : chr  "2019-08-22 01:00:16" "2019-08-22 01:00:16" ...
SNAP_ID            : int  35567 35567 35567 35567 35567 35567 35567 35...
INSTANCE_NUMBER    : int  1 1 1 1 1 1 1 1 1 1 ...
STAT_NAME          : chr  "active txn count during cleanout" "ADG pars...
VALUE              : num  5.11e+08 0.00 0.00 8.39e+07 1.19e+02 ...
```
- Import from export file integration
```
232897 obs. of  6 variables:
BEGIN_INTERVAL_TIME: chr  "2019-08-22 00:00:05" "2019-08-22 00:00:05"  ...
END_INTERVAL_TIME  : chr  "2019-08-22 01:00:22" "2019-08-22 01:00:22"  ...
SNAP_ID            : int  20070 20070 20070 20070 20070 20070 20070 20 ...
INSTANCE_NUMBER    : int  2 2 2 2 2 2 2 2 2 2 ...
STAT_NAME          : chr  "active txn count during cleanout" "ADG pars ...
VALUE              : num  94067174 0 0 24287453 0 ...
```
- Merged data table (long format) for analysis
```
114751 obs. of  4 variables:
snapHour  : POSIXct, format: "2019-08-22 00:00:00" "2019-08-22 00:00:00"  ...
```

```
STAT_NAME : chr  "ADG parselock X get attempts" "ADG parselock X get succ ...
sumValue_I: num  0 0 0 0 0 0 0 0 0 0 ...
sumValue_P: num  0 0 0 0 0 0 0 0 0 0 ...
```

Analysis Tools

- RGui 3.5.1
- Anaconda/Jupyter with Irkernel and R 3.6.0

Additional Comments

No considerations concerning CPU and memory consumption were made.

RISKS

No risks are known apart from those addressed by the data cleansing process. Outliers will be handled with during the data analysis process.

PRELIMINARY STUDIES

None

CONCLUSIONS

The analysis achieved its goal. It provided evidence that the integration database's performance is deteriorated in all cluster-related operations, when compared to the production database.

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