oTN-2008-01 openlab Technical Note

CERN PVSS tests on SAGE/Exadata

Anton Topurov, Eric Grancher
3 November 2008
Version 2
Distribution: Public

1 Introduction

PVSS is a commercial SCADA (Supervisory Control And Data Acquisition) system used for the supervision and control of the LHC experiments and some systems of the accelerator itself. PVSS is a product of ETM professional control GmbH, a Siemens Company. In general terms, PVSS is used to connect to hardware devices to send commands and configuration data to them, to acquire the data they produce and use it for their supervision and control, i.e. to monitor their behavior and initialize, configure and operate them.

1.1 PVSS Oracle Archiver

PVSS “Oracle Archiver” is the PVSS component, which takes care of storing and retrieving the real-time monitoring and control data in Oracle Database.

1.2 PVSS Oracle Archiver Workload

The workload of Oracle Archiver is mainly bulk insert of data into the database. Occasional queries are issued by the PVSS users. The LHC experiments have a requirement of insertion ratio of up to 150000 changes/s, which is unique case for SCADA applications usage.

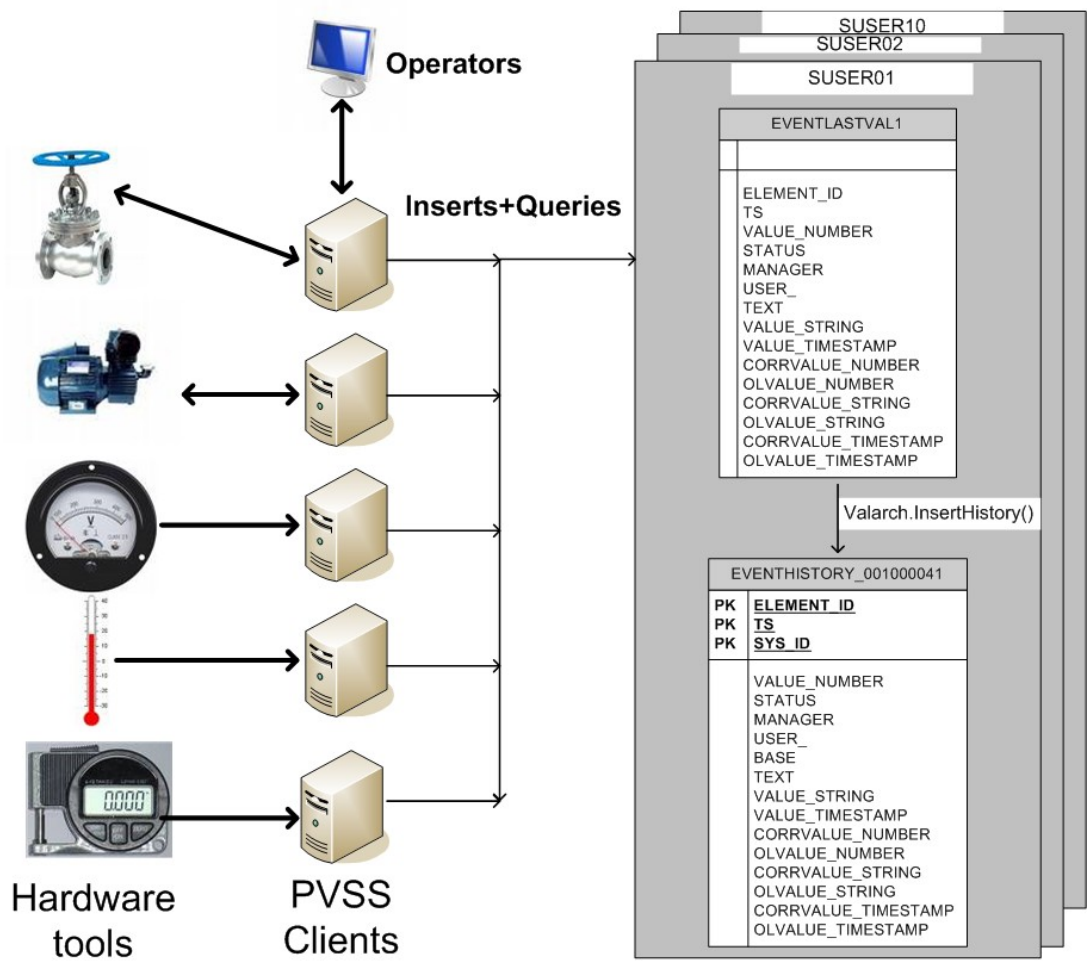

Figure 1 Schematic of PVSS workload

Generally the data is produced by the LHC and experiments' hardware, which is then acquired by PVSS clients. Data is buffered in the clients, and once sufficient amount is collected, it is inserted into the database, using Bulk Insert. There are several common-like schemas used to accumulate the data. Their structure is identical and comprise of a temporary table called EVENTLASTVAL (oracle special temporary table type), and permanent partitioned table called EVENTHISTORY_XXX (shortly EVENTHISTORY), where XXX is an archive number.
The data from PVSS is inserted into EVENTLASTVAL table.
Then, PL/SQL procedure InsertHistory takes care of transferring the data from the EVENTLASTVAL to a dedicated partition of the current active EVENTHISTORY table.
Once the tablespace, holding EVENTHISTORY table is almost full, a new tablespace is produced and new tables and indexes are created. During the creation of the tablespace, PVSS clients buffer the data locally. The total insertion performance (measured in changes/s) degrades. Moreover, depending on the total workload and the tablespace creation time, the buffers sometimes are not sufficient to hold the data, which leads to data loss.

2 SAGE advantage

SAGE is advertised as a smart storage, which can be used to offload some of the functionality of the database. In case of PVSS, offloading tablespace creation to the SAGE cell could be of great benefit, thus reducing the impact on PVSS clients loading and offloading the CPU / IO load from the database servers. If the impact of tablespace creation is too large, it can lead to the impossibility to load the data in time.

Currently SAGE hardware is not yet available for testing at CERN site.
Since PVSS set-up is very difficult to reproduce on the Oracle Site, we have created a PVSS workload-like benchmark, using Swingbench framework.

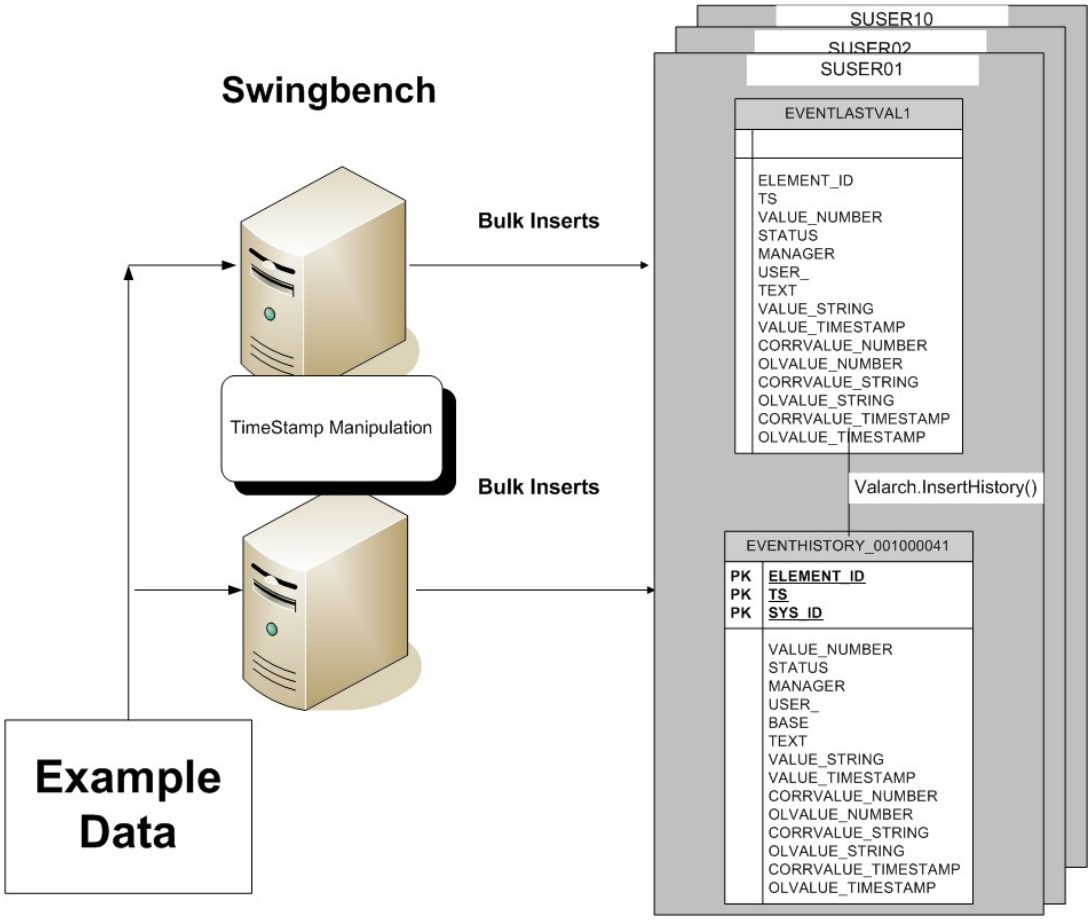

Figure 2 Swingbench PVSS benchmark

A CSV file with example data is read by our benchmark code, and the timestamp of the records is changed to be close to the SYSTIMESTAMP. The actualized records form a bulk insert statement, which is executed every second, thus simulating 1000 changes/s load. After each bulk insert our code calls the InsertHistory procedure, which transfers the data to the needed table partition.
Every 7 minutes a job checks the size of the tablespaces, and once they are above a certain threshold, the tablespace switch is initiated. In addition our code logs execution time of Bulk inserts into specific table, which is used for detailed analysis of the benchmark results.

3 CERN test system

3.1 Setup

Two-instance RAC

- 2 CPU Intel(R) Xeon(R) 5140@2.33GHz (4 cores in total)
- 8 GB RAM (sga_target =6GB)
- NAS based storage.

3.2 Test Results

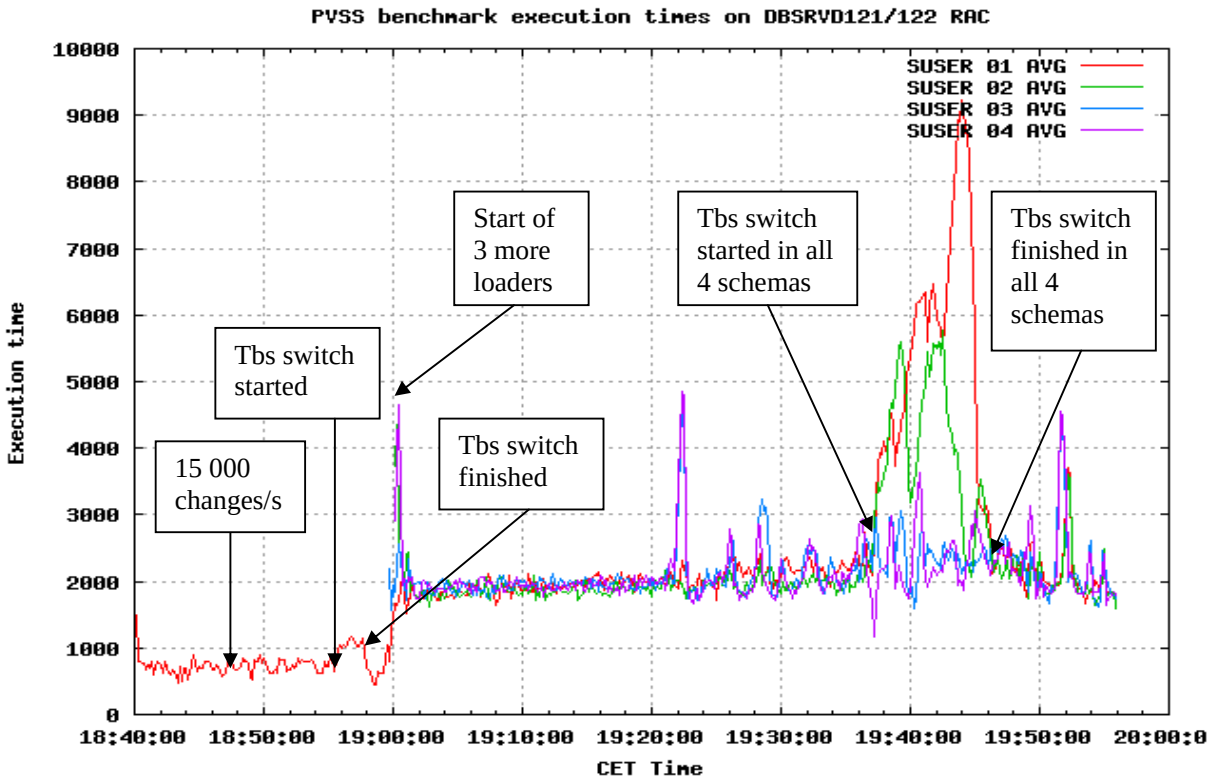

Figure 3 CERN test system results

1000 ms execution time is a buffering threshold, i.e. higher execution time means that the system can not cope with the load and started buffering.
At 18:40 there was 15 sessions running on the instance 1 of the database (load 15 000 changes/s). Execution times were below 1000ms, until the tablespace filled up and the tablespace switch started at 19:55. During the tablespace creation the “buffering” was light and once the switch happened, the execution times normalized.

At 19:00 additional 45 sessions were started, making the desired load of 60 000 changes/s. The graph shows, that the system could not cope with the load and was buffering.

At around 19:36 multiple tablespace switch started – i.e. each of the 4 schemas were doing the switch. The execution times were too high, indicating that such a load is not possible to cope with.
Once the switches finished, the execution times went back to around 2000 ms level.

Conclusion: Tablespace switch is too costly for the system.

4 SAGE test system

4.1 Setup

Four-instance RAC

- XXX CPU
- 16 GB RAM (sga_target =10GB)
- 4 SAGE cells storage

4.2 Test results (16th of September)

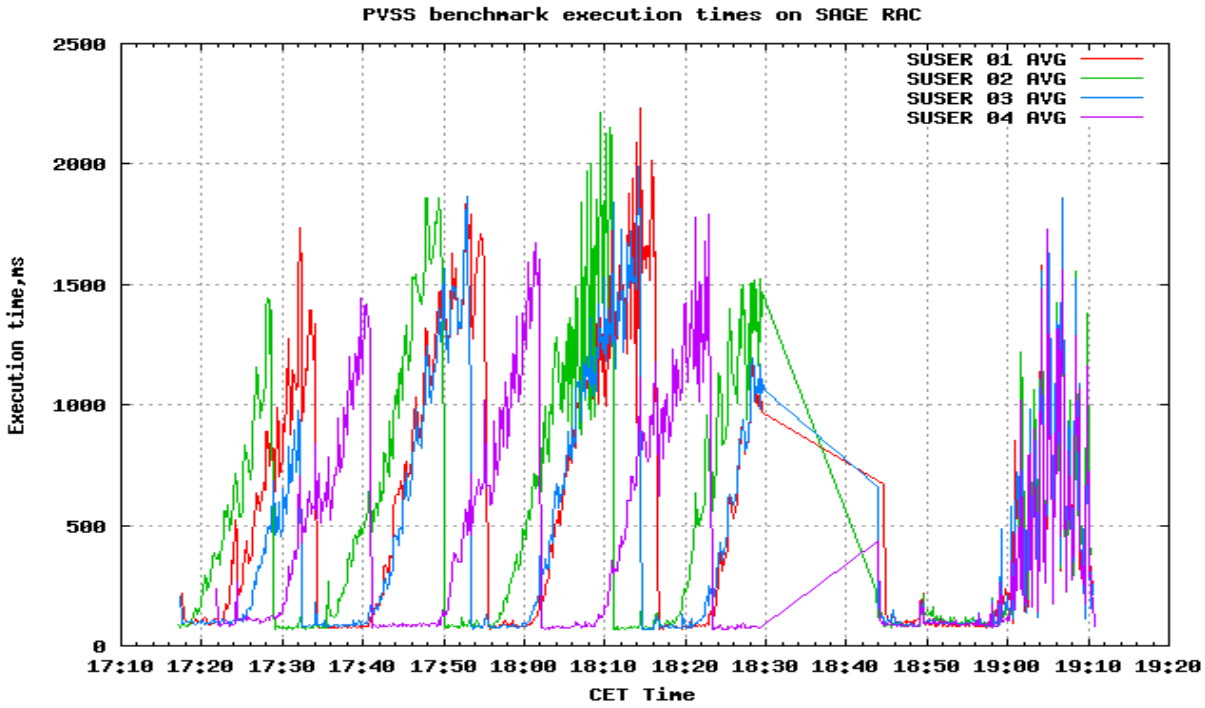

Figure 4 SAGE test system results

For simplicity of the analysis, we will use the graph of only two schemas.

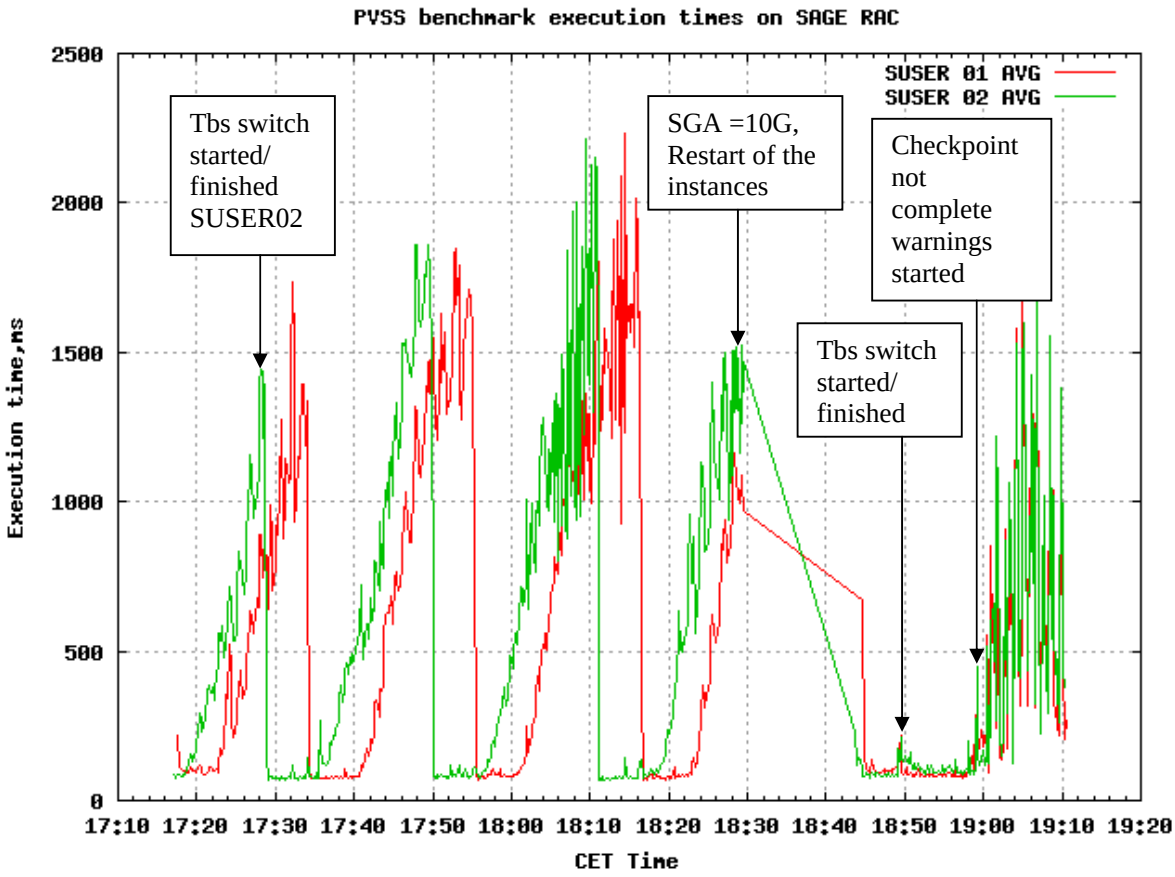

Figure 5 SAGE test system results (two schemas)

The tests were started around 17:20 UK Time.
Since sga_target was 1GB, the execution time was increasing due to fill of cache (mainly indexes were causing that).
As soon as tablespace switch happened, the new table and its respective index were created; execution times were back to normal.
Around 18:30 the sga_target was set 10 GB, followed by restart of the instances at 18:41.Performance have improved (the execution times stayed low for longer periods of time).
At 18:59 we started 4 more swingbench instances (SUSER05-08), which lead to slight decrease in performance. This can visible in the very shaky part of the graphics. “Checkpoint not complete” was major message in the alert logs, indicating that redo log files have not sufficient size for the load.

19:10 – we stopped the tests, since other testers were waiting for the machines.

4.3 Test results (22-23 of October)

Since the announcement at Oracle Open World of Oracle Exadata, the word Exadata will be used instead of SAGE.

October 23rd Tests

System Parameters:
Sga_target = 10G

Pga_aggregate_target=10G
Db_block_size=8K
Event Tablespaces = ***3GB***, ***5M*** uniform size

Description: The test intended to find out if the system was able to cope with the 150 000 changes/s load. The tablespaces size was set to 3GB, like in September tests. The setup of the system was done during the first half of the day; the actual tests were started after 14:00. The load increase was done with 15 000 changes/s steps.

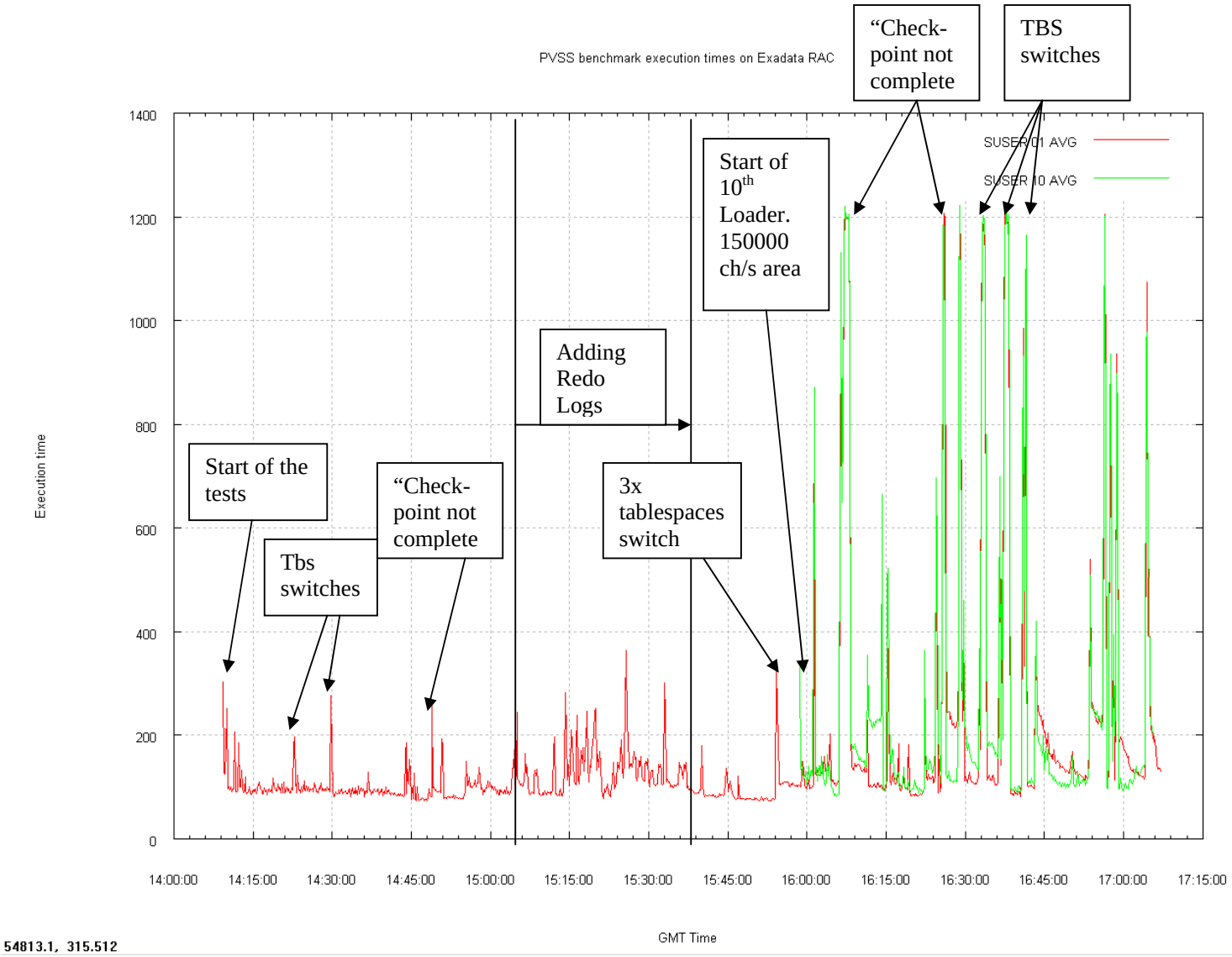

Figure 6 Execution time graph of the first day

Increasing the load from 0 to 120 000 went smoothly, and no buffering occurred. "Checkpoint not complete" messages appeared in the alert.log, so we have added 2 more redo logs per instance, 4GB each. Moving from 120 000 changes/s to 135 000 went ok.
Once we tried to reach 150 000/s (the green line starts exactly at that time), the execution times more then tripled, crossing the buffering threshold of 1000 ms execution time.

Analysis of AWR and wait events showed contention on write on control files.
During this period any “Checkpoint not complete” or tablespace creation leaded to execution times, which were greater then the “buffering” threshold.
We were not sure if the double and triple simultaneous tablespace creation was the trigger for the instability, therefore we have decided to use 20GB tablespaces, which would give us ~2 hours of “tablespace switch free” time for analysis.

October 24th Tests

System Parameters:
Sga_target = 10G
Pga_aggregate_target=10G
Db_block_size=8K
Event Tablespaces = ***20GB***, ***5M*** uniform size, later ***16MB*** uniform size

Description: During this day we tried to identify the maximum point of stability and try to achieve 150 000 changes/s using 20GB tablespaces. Changing uniform size of the tablespaces to 16MB was suggested by Oracle as mean to improve performance.
In addition we tried to quantify the performance gain given by the fast file creation feature of Exadata, by switching it off and on.

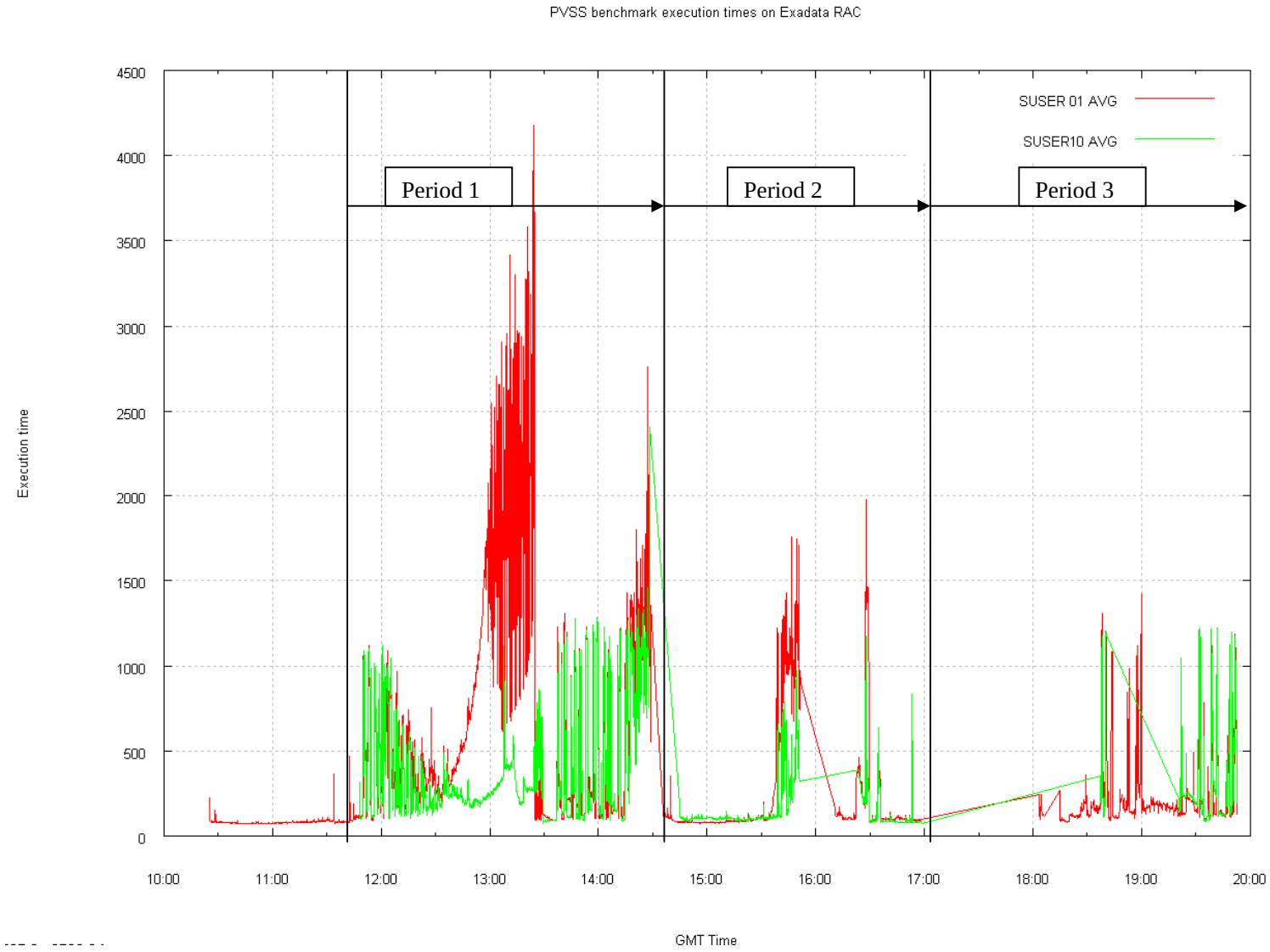

Figure 7 Execution time graph of the second day.

The graph is divided in 3 periods, which will be shown and analyzed separately in this report.

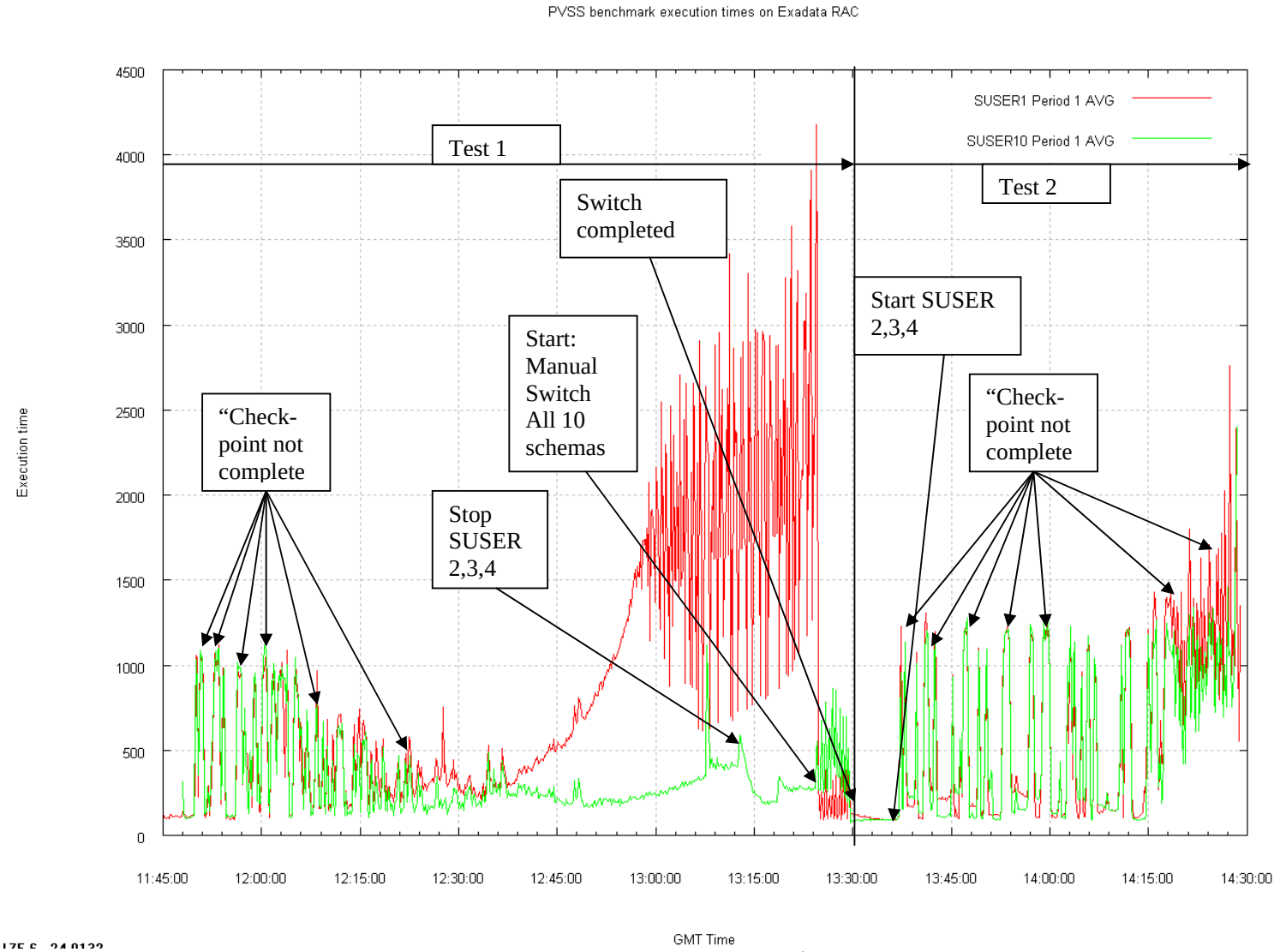

Figure 8 Execution time graph, day 2, 1st Period

Test 1: This test uses 20GB tablespaces, so the tablespace switch occurs not that frequent as during the previous day. At 11:45 we have achieved a stable load of 120 000 changes/s. Then we have started additional 30 000 changes/ s, totalling in 150 000 changes/s. This load showed very little instability in the beginning (some of the exec times were bigger then 1000), and then recovered. After one hour, the execution times became much higher, and no recovery appeared by itself.
At 13:10 we have stopped 3 of the loaders (removed 45 000 changes/s load), but this did not help.
The sum size of used extents in the tablespace was around 9G, while the automatic switch should appear when sum extent size is higher then 15G. Therefore we forced manual switch of the tablespaces at 13:24.

Test 2: To double check the previous test, we left the system running at 105 000 changes/s for couple of minutes, which was stable. Then we started back the 3 swingbench instances, which were stopped during the previous test. This lead to execution times around and above the threshold of 1000 ms. Performance did not recover, and after 1 hour we decided to stop it. This test confirmed the behavior of the Test 1.
Most of the peaks in the execution time graph correlated "Checkpoint not complete" message in the alert.log

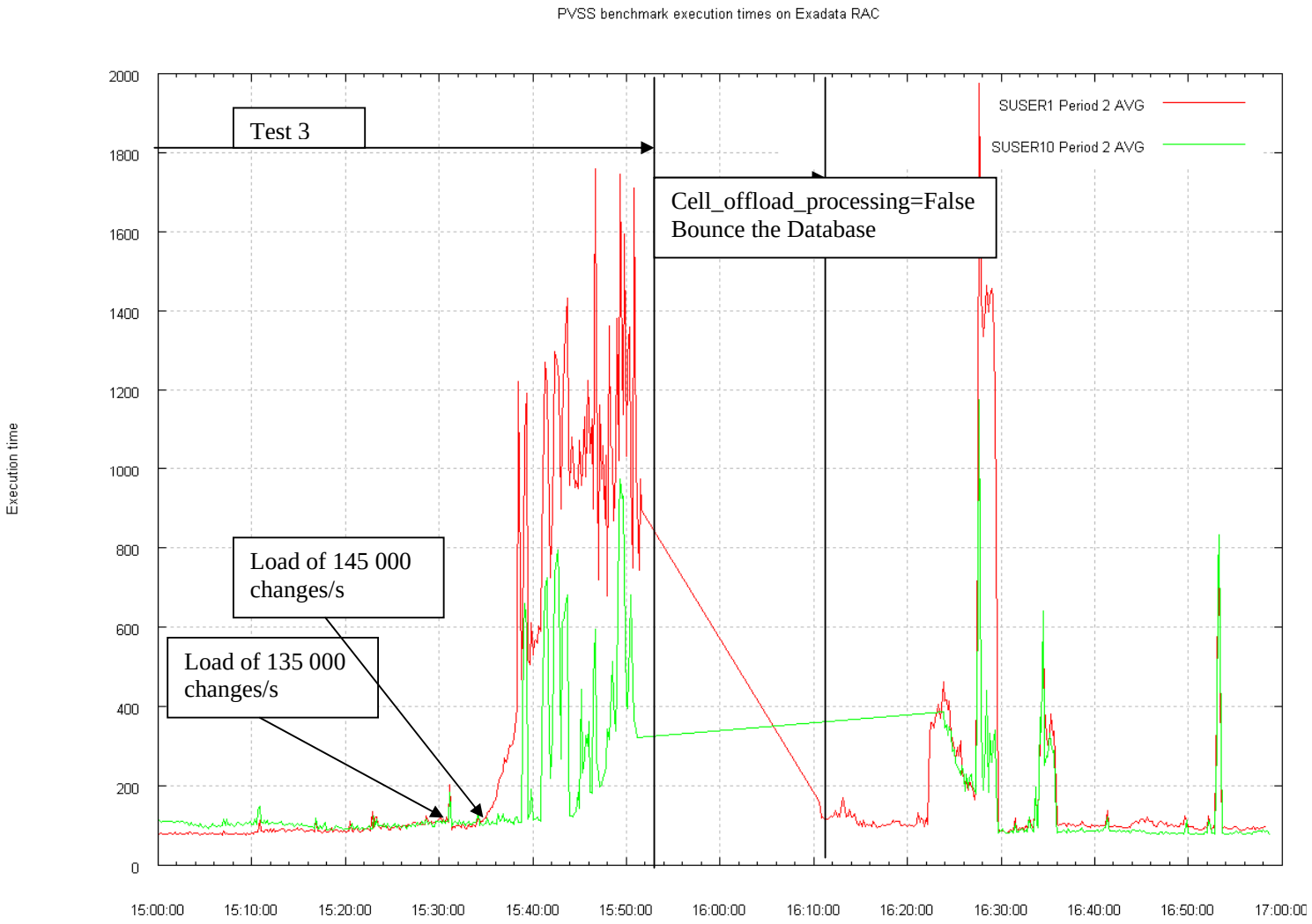

Figure 9 Execution time graph, day 2, 2nd Period

Test 3: The idea of the test was to find the maximum stable point.
We have started from 0 and reached 120 000 changes/s with no problem. Then we increased the load with the step of 1000 and 2000 changes/s second. The execution time degrades at 145 000 changes/s, at around 15:36.

Next, with the limited time left, we wanted to quantify the effect of the fast_file_creation feature.
We put Cell_offload_processing = off, expecting that this will disable fast file creation as well, but this was not the case, as 20GB tablespace was created in 17 seconds, as before. The big spike at 16:27 is due to too big load put by mistake.

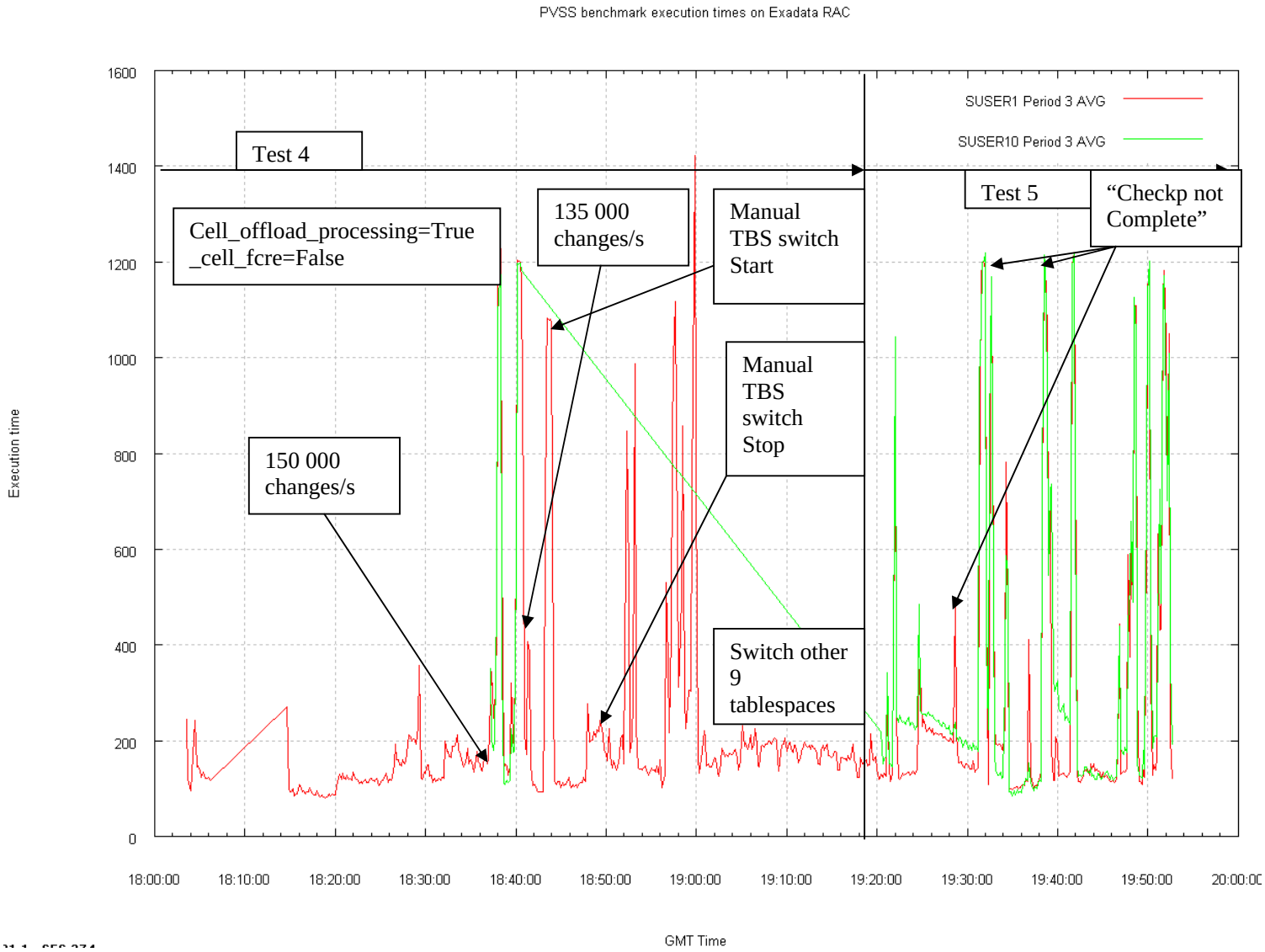

Figure 10 Execution time graph, day 2, 3rd Period

Test 4: We have found an additional parameter _cell_fcre, which was supposed to disable only the fast file creation. We have set back the cell_offload_processing to TRUE. There were problems during the bounce of the database, but finally it recovered and we could continue with the next test.
150 000 load was problematic again, and we reduced the load to 135 000, to see the effect of tablespace switch, when the fast file creation is disabled.

Indeed, during the creation of the new tablespace, there was a spike in the execution time. However, when we did switch to other 9 tablespaces, we did not observe such strong spikes.

Test 5: During the last switch, we have created the tablespaces with 16MB uniform size, which was advised by Oracle engineers. This gave a bit better result – but not sufficient to sustain for a long time.

5 Conclusions

PVSS data loading to a database is a very important operation. The data loading performance suffers from the necessary tablespace creation operations. These tablespace creation operations lead to degradation of performance (i.e. buffering and potential data not being loaded and possibly lost).

In a test system we could sustain a load of 30 -35 000 changes/s, with insertion time going very high during tablespace creation.

In Exadata environment, we measured much better results, thanks to the strong / balanced capacity of the nodes and storage and the Exadata Storage Server fast file creation feature.

We could achieve a stable rate of 145 000 changes/s. Change from 5MB uniform size to 16MB uniform size improved stability a bit, but could not totally solve the problem. The top wait event was “concurrent change of control file”, which we could not eliminate. Oracle engineers suggested reducing the number of control files, but there was no time left for this test.

Tablespace creation with “fast file creation” took approx 17 seconds for 20GB tablespace, compared with ~2 minutes with this feature switched off. This resulted in much smaller spikes in the execution times in comparison with the spike while the feature was switched off. This establishes the proof of the benefit of the feature in such a workload.