Analysis of possibilities to increase the efficiency of the relative database management system using the methods of parallel processing

Bronisław Wajszczyk*
Ireneusz Marek Gruszka**
Institute of Radioelectronics, Military University of Technology
Gen. S. Kaliskiego str. 2, 00-908 Warszawa, POLAND

ABSTRACT

This article presents the results of research on improving the relational database performance through the use of software and hardware parallel processing methods. The research has been based on the Oracle Relational DataBase Management System, which is one of the few to perform parallel queries. The choice of Oracle to use as a base for the system has been dictated by its capabilities, namely support for Linux (Linux is widely used in radio-electronic systems used for military applications), support for most of the used indexes, and support for various operations and security used in RDBMS.

Keywords: data base, parallel processing, methods of data access optimization

1. INTRODUCTION

Data access time is a critical parameter in radio-electronic systems. Examples of solutions for radio-electronic systems in which access to data is a critical parameter have been described in the articles [9,10,16,17]. These solutions are associated with the creation of microwave signal patterns in radio-electronic reconnaissance devices. This applies primarily to radio-electronic reconnaissance systems and devices, where measurement results are compared with the patterns stored in the database. That comparison is expected to take place in a time similar to real time. This requires searching for efficient methods of accessing data stored in databases. One of the solutions is to use parallel processing techniques which utilize the computing power of available hardware resources more efficiently. The problem of parallel processing in databases is raised in many publications. In paper [12] the problem of classification for Architecture of Parallel Databases was raised. Other papers presented query optimization for parallel processing in databases. [11,13,14,15].

2. METHODS AFFECTING THE DATABASE SERVER PERFORMANCE

Methods of performance improvement can be divided into software and hardware. Programming methods include the possibility of using indexes on tables, partitioning tables and indexes, maintaining statistics for data contained in database objects, automatic tuning of query plans, parallel processing, buffering. Hardware methods include the use of many physical computational clusters, skilful distribution of data files, and fine-tuning of the RDBMS system operation to maximize the processor's capabilities and appropriately allocated memory [2].

2.1 Software methods

In order to speed up data retrieval in tables, RDBMS uses various types of indexes [1]. The author's work [8] discusses the basic indexes and the principles of their use. Partitioning involves splitting the table into partitions according to system requirements, e.g. partitioning of the microwave patterns database into partitions according to the adopted frequency range. Partitioning allows to increase the performance of queries that reference one partition instead of searching the entire table [1].

Tuning in the RDBMS system is done using the native Procedural Structured Query Language (PL / SQL). In the DBMS_PROFILER package RDBMS stores procedures, which offer a number of actions to analyze and suggest improvement of the application code. With this package, one can easily identify the most expensive operations in the code and try to optimize their operation.

*bronislaw.wajszczyk@wat.edu.pl; phone + 48 261 837 062; fax + 48 261 837 461; www.wat.edu.pl
**gruszka.ireneusz@gmail.com

Radioelectronic Systems Conference 2019, edited by Piotr Kaniewski, Jan Matuszewski, Proc. of SPIE
Vol. 11442, 1144215 · © 2020 SPIE · CCC code: 0277-786X/20/$21 · doi: 10.1117/12.2565744

2.2 Parallel processing

Each query in the database is analysed by the optimizer (parsing). If the execution plan includes parallel processing, the following steps occur [5]:

- the background or session process assumes the role of query coordinator,
- the coordinator collects information about the number of processes that a given query has at its disposal,
- the query is carried out sequentially, various operations are carried out, if possible, there are several of these operations and are performed simultaneously
- when the parallel processes are completed, the coordinator starts to perform tasks that could not be started in parallel,
- when the job is complete, the coordinator returns the results of the queries to the user.

After arranging the query plan, the parallel execution coordinator - (PX coordinator) decides how the subtasks will be executed. The number of processes that can be started for one task is defined by Degree Of Parallelism (DOP). The DOP level can be defined by the user or automatically by the RDBMS system.

3. RESEARCH ENVIRONMENT

The purpose of the studies was to assess DOP that should be used to achieve maximum query performance in the test conditions. Furthermore, there has been made an attempt to answer the questions: - whether the use of parallel queries significantly affects the performance of the database management system, - in what hardware and software conditions the use of parallel processing is beneficial from the point of view of RDBMS system performance.

The research has been carried out for various measurement conditions that included: the operating system used (Linux and Windows), a different number of processor cores, a different number of parallel sessions, the use of efficient SSDs, the use of the index. Two test environments have been prepared for the purposes of the study,

Research Environment no. 1

- Machine: Notebook DELL XPS L702X,
- RAM memory: 8GB DDR3,
- CPU: 4 x CPU Intel Core i7-2670QM 2.20GHz,
- HDD: SSD disk 180GB and HDD 500GB,
- Operating system: Windows 7 SP1 64bit.

Research Environment no. 2

- Machine: Virtual server on the VMware ESX 5.1 platform,
- RAM memory: 8GB,
- CPU: 8 x CPU Intel Xeon E5-2670 2.6GHz,
- HDD: Two logical volumes of 1TB were separated from the matrix,
- Operating system: Linux CentOS 6.4 64bit.

Oracle Enterprise Edition 11.2.0.4 64-bit has been selected for the research. The test database environment has been prepared on the basis of scripts and tips, available in the article "How Many Slaves?" by well-known Oracle database administration specialist Doug Burns [6]. The test database was prepared based on the scripts for the RDBMS study published by Oracle [7]. Modifications have been introduced, especially in the scope of script preparation for the Windows environment. The parameters of the database instance are shown below,

```
db_cache_size=1258291200
testmgr.__java_pool_size=16777216
testmgr.__large_pool_size=335544320
testmgr.__oracle_base='G:\app\oracle'#ORACLE_BASE set from environment
testmgr.__pga_aggregate_target=671088640
testmgr.__sga_target=2013265920
testmgr.__shared_io_pool_size=0
testmgr.__shared_pool_size=385875968
testmgr.__streams_pool_size=0
*.audit_file_dest='G:\app\oracle\admin\testmgr\adump'
*.audit_trail='db'
*.compatible='11.2.0.4.0'
```

```
*.control_files='C:\oracle\oradata\testmgr\control01.ctl',
'G:\app\oracle\fast_recovery_area\testmgr\control02.ctl'
*.db_block_size=8192
*.db_domain=''
*.db_name='testmgr'
*.db_recovery_file_dest='G:\app\oracle\fast_recovery_area'
*.db_recovery_file_dest_size=4385144832
*.diagnostic_dest='G:\app\oracle'
*.dispatchers='(PROTOCOL=TCP) (SERVICE=testmgrXDB)'
*.job_queue_processes=0
*.open_cursors=300
*.parallel_adaptive_multi_user=FALSE
*.parallel_max_servers=512
*.pga_aggregate_target=671088640
*.processes=800
*.remote_login_passwordfile='EXCLUSIVE'
*.sga_target=2013265920
*.undo_retention=600
*.undo_tablespace='UNDOTBS1'
```

The database instance parameters that apply to parallel processing are listed below.

```
SQL> show parameters parallel

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
fast_start_parallel_rollback         string      LOW
parallel_adaptive_multi_user         boolean     FALSE
parallel_automatic_tuning            boolean     FALSE
parallel_degree_limit                string      CPU
parallel_degree_policy               string      MANUAL
parallel_execution_message_size      integer     16384
parallel_force_local                 boolean     FALSE
parallel_instance_group              string
parallel_io_cap_enabled              boolean     FALSE
parallel_max_servers                 integer     512
parallel_min_percent                 integer     0
parallel_min_servers                 integer     0
parallel_min_time_threshold          string      AUTO
parallel_server                      boolean     FALSE
parallel_server_instances            integer     1
parallel_servers_target              integer     128
parallel_threads_per_cpu             integer     2
recovery_parallelism                 integer     0
```

It is worth paying attention to the following parameters:

- **parallel_adaptive_multi_user** - launches a mechanism that blocks the use of all available resources when using parallel processing,
- **parallel_io_cap_enabled** - causes parallelization level limitation due to given system parameters,
- **parallel_max_servers** - specifies the number of processes that the database instance can start to handle queries that use parallel processing.

Eight tables have been created for the purposes of the study: TEST_TAB1 - TEST_TAB8, where:

- TEST_TAB1, TEST_TAB2 - tables with 8 million rows,
- TEST_TAB3 to TEST_TAB8 - tables with 128 thousand rows.

The script for creating the TEST_TAB1 table and filling them with rows with random data - universal for both operating systems is presented below:

```
connect testuser/testuser
set echo on
set lines 160
alter session set recyclebin = off;
alter session set max_dump_file_size=unlimited;
alter session enable parallel dml;
alter session enable parallel ddl;
execute dbms_random.seed(999);
create sequence test_seq1 cache 1000;

REM creates a table of 1000 rows, a primary key based on sequences, repeating values from 1 to 10 and a random number
create table test_tab1
pctfree 90 pctused 10
as select test_seq1.nextval PK_ID, MOD(test_seq1.nextval, 10) NUM_CODE,
dbms_random.string('A', 100) STR_PAD
from all_objects
where rownum <= 1000;
REM Increasing the number of poems to 8 million.
begin
for i IN 1 .. 13 loop
insert /*+ parallel(test_tab1, 2) append */
into test_tab1
select /*+ parallel(test_tab1, 2) */
test_seq1.nextval, NUM_CODE, STR_PAD
```

```
from test_tab1;
commit;
end loop;
end;
/
REM The conversion of statistics for the CBO cost optimizer.
begin
dbms_stats.gather_table_stats(user, 'test_tab1', cascade => false,
estimate_percent => 1,
method_opt => 'for all columns size 1');
end;
```

The script below creates the TEST_TAB3 test tables:

```
connect testuser/testuser
set echo on
set lines 160
spool setup3.log
alter session set recyclebin = off;
alter session set max_dump_file_size=unlimited;
alter session enable parallel dml;
alter session enable parallel ddl;
execute dbms_random.seed(999);
create sequence test_seq3 cache 1000;
create table test_tab3
pctfree 90 pctused 10
as select test_seq3.nextval PK_ID, MOD(test_seq3.nextval, 10) NUM_CODE,
dbms_random.string('A', 100) STR_PAD
from all_objects
where rownum <= 1000;
begin
for i IN 1 .. 7 loop
insert /*+ parallel(test_tab3, 2) append */
into test_tab3
select /*+ parallel(test_tab3, 2) */
test_seq3.nextval, NUM_CODE, STR_PAD
from test_tab3;
commit;
end loop;
end;
begin
dbms_stats.gather_table_stats(user, 'test_tab3', cascade => false,
estimate_percent => 1,
method_opt => 'for all columns size 1');
end;
```

Four tests have been developed for research:

- **Test No. 1** - analysis of the impact of using parallel processing with a given server configuration - a variant with two large tables. Queries consisting of full table scanning and hash joining with data grouping have been examined. The test has been carried out in two variants: for Linux and Windows. On Linux it was possible to change the number of processor cores connected. Both variants controlled the DOP value between 1 and 128. The following is a SELECT query that is included in the test script:

```
SELECT /*+ parallel(tt1, %DOP) */ COUNT(*)
FROM test_tab1 tt1;

SELECT /*+ parallel(tt1, %DOP) parallel(tt2, $DOP) */ MOD(tt1.pk_id + tt2.pk_id, 113), COUNT(*)
FROM test_tab1 tt1, test_tab2 tt2
WHERE tt1.pk_id = tt2.pk_id
GROUP BY MOD(tt1.pk_id + tt2.pk_id ,113)
ORDER BY MOD(tt1.pk_id + tt2.pk_id ,113);
```

- **Test No. 2** - analysis of the impact of using parallel processing with more parallel sessions - a variant with a large and smaller table. The tests have been carried out for the TEST_TAB1 table and one of the smaller TEST_TAB tables (3-8), which have been selected randomly based on the number of sessions given in the parameter. The choice of the smaller table has been determined by the formula: TEST_TAB = (<SESSION_SESSION> MOD 8) + 3, Test No. 2 was conducted in environment No. 2 in the scope of parallel sessions from 1 to 64, in the number of processor cores from 1 to 8 and DOP from 1 to 128, with the average query time for all DOP levels taken into account. The following is the SELECT query that was used in the test script:

```
SELECT /*+ parallel(tt1, %DOP) parallel(tt2, $DOP) */ MOD(tt1.pk_id + tt2.pk_id, 113), COUNT(*)
FROM test_tab1 tt1, test_tab3 tt2
WHERE tt1.pk_id = tt2.pk_id
GROUP BY MOD(tt1.pk_id + tt2.pk_id ,113)
ORDER BY MOD(tt1.pk_id + tt2.pk_id ,113);
```

- **Test No. 3** - analysis of the impact of using the index on the time of reading data from the largest table and comparing the results with the parallel processing technique. The study took place in environment No. 2 using the most efficient processor configuration - with 8 connected cores. For the purposes of the study, an index was created for the table TEST_TAB1 named TT1_STRNAME_INDX. The index creation command and test query is as follows:

```
create index TT1_STRNAME_INDX on TEST_TAB1 (STR_PAD)
  tablespace TEST_DATA
```

```
  pctfree 10
  initrans 2
  maxtrans 167
  storage
  (
    initial 1M
        next 1M
    minextents 1
    maxextents unlimited
    pctincrease 0
  );

# The CBO optimizer will automatically use the index TT1_STRNAME_INDX
SELECT /*+ parallel(tt1, $DOP) */ COUNT(*)
FROM test_tab1 tt1
where STR_PAD like 'c%';

# Index use will be blocked by forcing no_index
SELECT /*+ parallel(tt1, $DOP) no_index(tt1) */ COUNT(*)
FROM test_tab1 tt1
where STR_PAD like 'c%';
```

– **Test No. 4** - analysis of the impact of using an efficient SSD drive on the query read time and correlation of this change with the parallel processing technique. The study has been conducted in environment No. 1 using 8 processor cores. Two types of queries were tested: full table scan and hash join of the two largest tables TEST_TAB1 and TEST_TAB2. The tests have been performed first using a high-performance SSD and then using an HDD. The influence of the speed of read / write operations on the time of response was examined. In addition, response time studies were conducted using different DOP levels in the range of 1 to 128.

4. RESEARCH RESULTS

The level of parallelization has been tested in the range from 1 to 128 with the following step: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 24, 32, 48, 64, 96, 128. This allowed the identification of differences in query performance at a low level of parallelism, where these differences were most visible, and the observation of the impact of the overhead associated with running a very large number of processes at a level of parallelization of over 100. The level of parallelism recorded on the horizontal axis of charts as 1 means no parallel processing applications, and thus no additional subprocesses (Parallel Execution Slaves - PX Slaves) have been created.

4.1 Test No. 1 - CentOS system

The test results are presented in Fig. 1. The data series in the graph determine the application of the server configuration with a given number of processor cores in the range from 1 to 8.

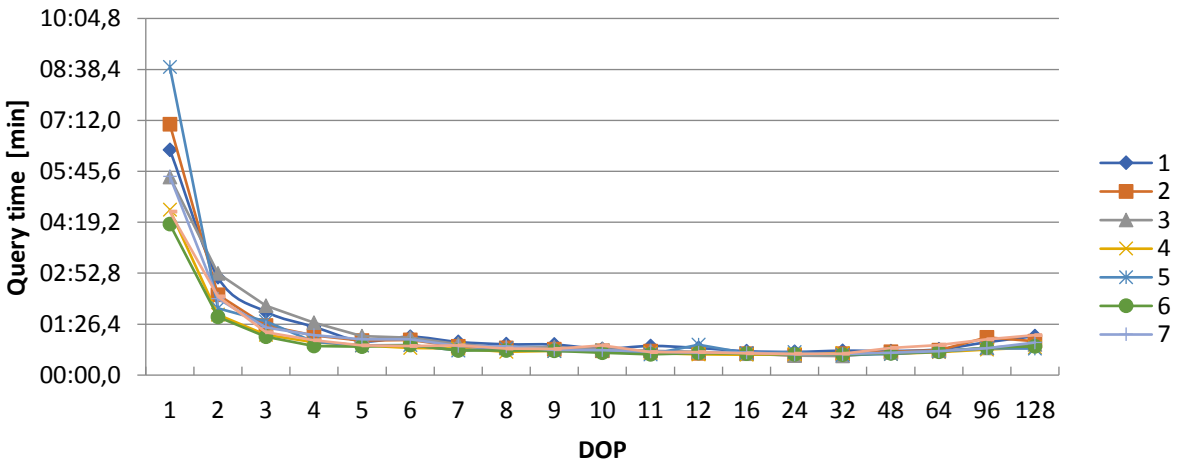

Figure 1. Query times for different number of processors for full table scan operations (CentOS system)

Conclusions resulting from the conducted research:

– the use of parallel processing brings a significant improvement in query performance with full table scanning,

– the decrease in query execution time is the largest between a DOP level of 1, i.e. without parallel processing, and a DOP level of 2,
– the largest decrease in the above range was observed with 5 processors, which is due to the fact that the configuration of the virtual machine allowed the processor cores to be added to one or two virtual sockets,
– a noticeable increase in query performance can be seen up to a level of parallelization of 3, the percentage increase is still small, and even starts to fall at a DOP level of 48.

It is important to notice that during hash joining of tables with grouping and sorting commands, twice as many subprocesses - PX Slaves have run than in the case of full scan. That results in much longer response times, as shown in Figure 2.

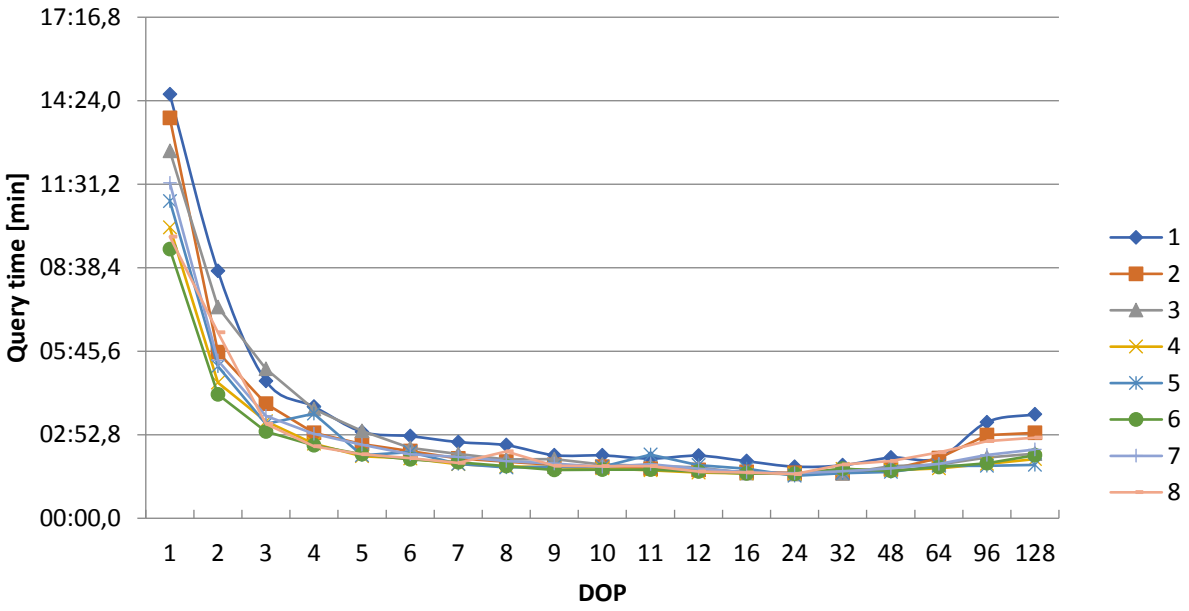

Figure 2. Query times for different number of processors for hash joining operations (CentOS system)

Conclusions resulting from the conducted research:

– the use of parallelism no longer brings such a performance improvement as in the case of full table scanning, which is caused by a greater level of complexity of the performed operation and the need to run additional processes by the coordinator,
– however, there is a greater improvement between levels 2 and 3, and 3 and 4. Only at DOP levels above 5 the "flattening" of the line can be seen on the chart, which suggests that the use of a higher level of parallelism for a more complicated operation results in an increase in the overhead associated with the need to support a larger number of processes, and thus no further improvement of query performance can be achieved,
– above the DOP level of 32, query performance drops.

Figure 3 shows the CPU usage (s) during the test. The start of testing is marked with a vertical black line. The first range of processor activity (to the right of the vertical line) covers the testing time using a single-core processor. One core was added successively. It can be seen that only for the configuration with a single-core processor, the resources have been used in 100%, which basically means that at that time any other work on the server would have been practically impossible. In other cases, the processor's idle time has been greater than 0, so it can be assumed that the server would have been able to handle more parallel sessions.

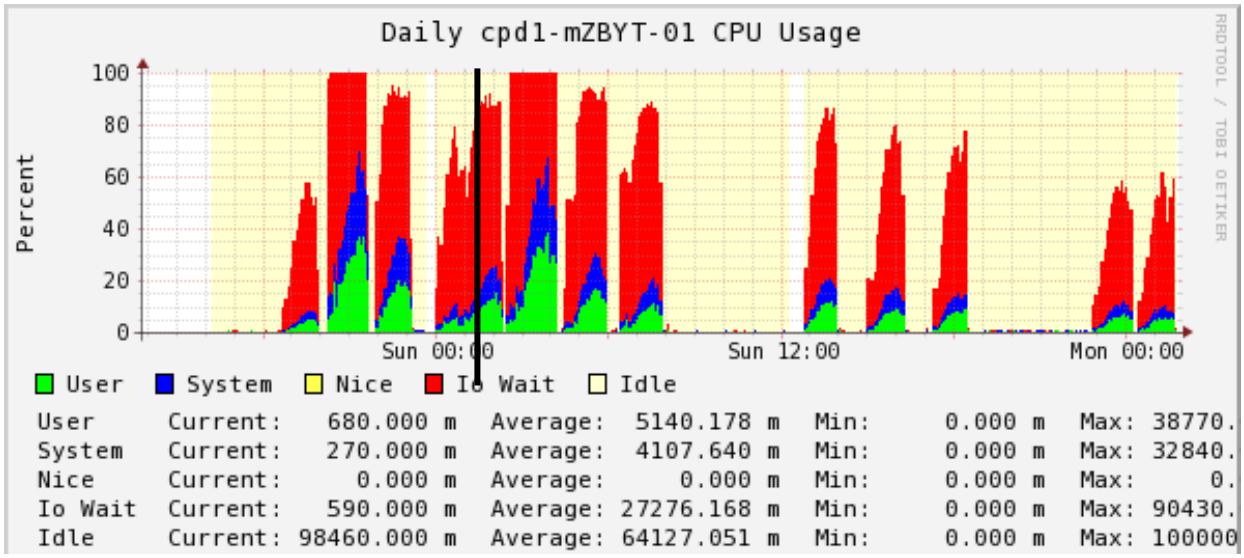

Figure 3. Characteristics of the processor's work - percentage consumption during testing as part of Test No. 1 in the CentOS system

The red horizontal line in Figure 4 indicates the number of processor cores available, and above it the processes running in the queue have been marked for each configuration, which in practice means that each processor core must perform several dozen operations in series.

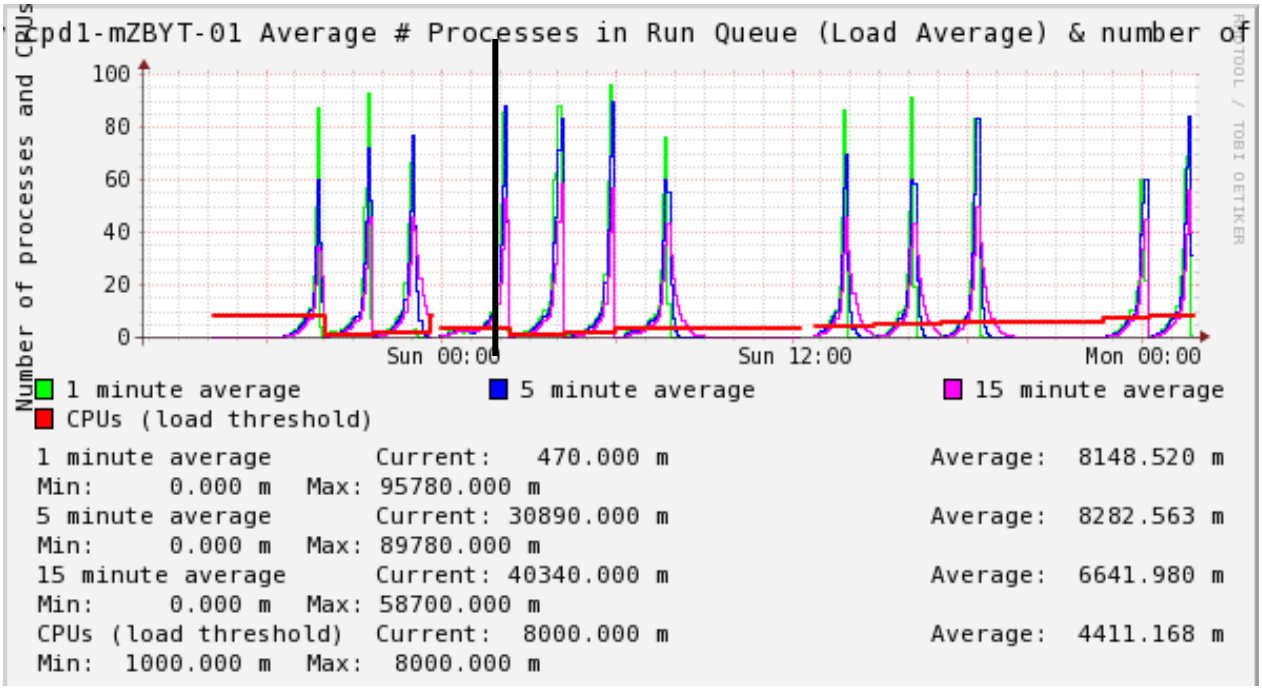

Figure 4. The number of processes in the processor's input queue during testing as part of Test No. 1 in the CentOS system

Analysing Figure 5, it can be concluded that increasing the number of processor cores does not affect the operation of disks, which must be used equally to return the desired result.

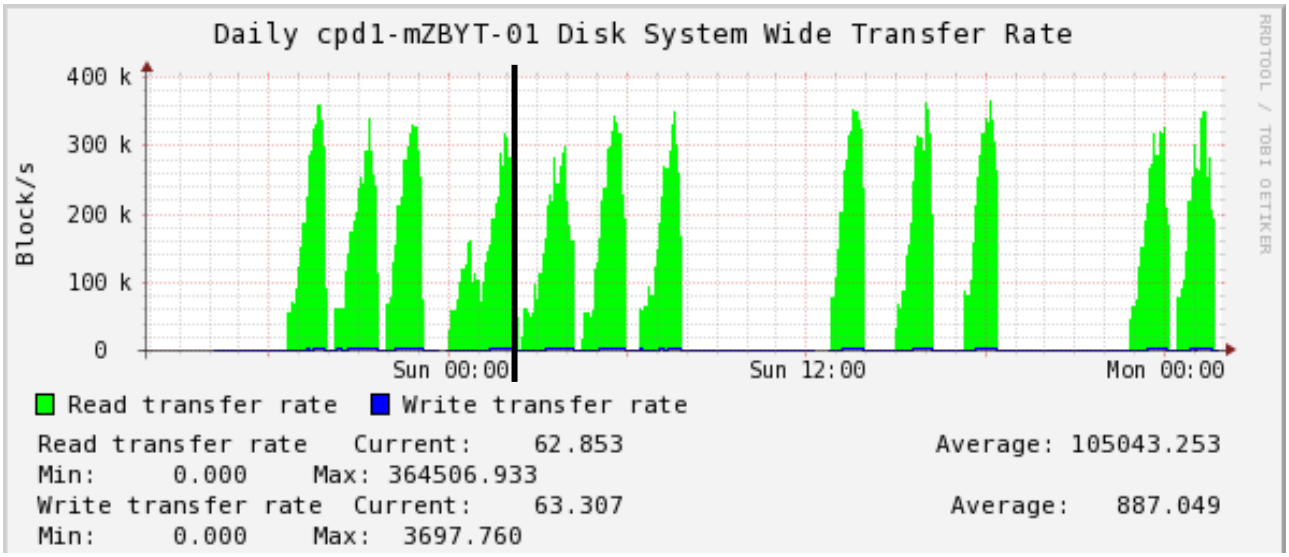

Figure 5. Disk operation characteristics - data transfer coefficient in blocks per second in the CentOS system

4.2 Test No. 1 - Windows system

The test results are shown in Figure 6.

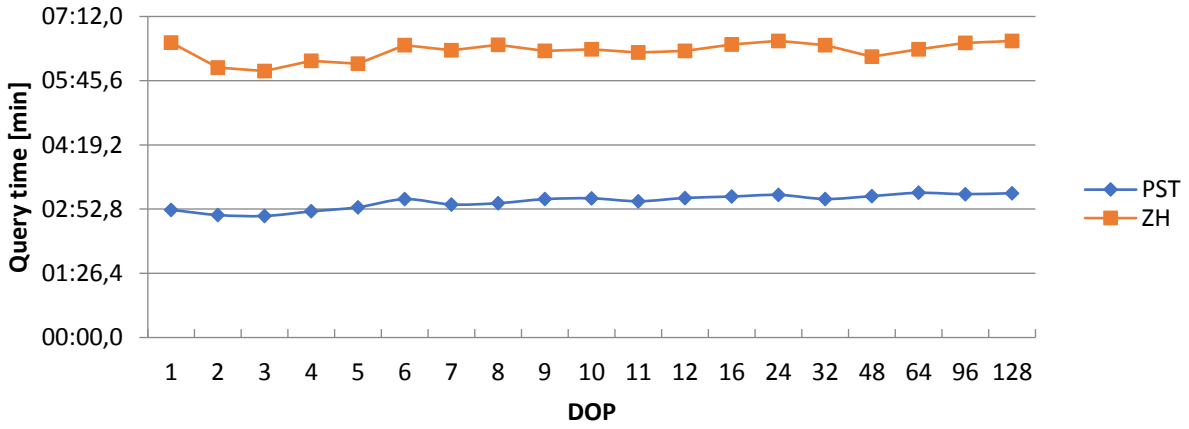

Figure 6. Query times for full table scan operations (8 processor cores) and table hash joining (Windows)

Figure 6 shows query times for both the full table scan operation - the PST series and the hash join - the ZH series. Conclusions resulting from the conducted research:

- query times for hash joining operations are higher, which is simply a result of their complexity,
- the introduction of parallel processing increases the efficiency of operations by reducing the time it takes to execute the query in both cases, however, this increase is not as significant as in the case of tests in the CentOS system, which may be due to the fact that the computer works more stable and the readings take place only on one disk HDD, not physically on several different disks of the matrix, as in the case mentioned above,
- weaker CPU resources (1 quad-core processor with dual-threading capability) mean that parallelism brings much less benefits than a more powerful Linux server.

4.3 Test No. 2 - CentOS system

The results of tests for test No. 2 are presented below. During the tests, the processor configuration was changed in the range of 1 to 8 cores. Query times for different sessions in most cases have been identical or have not differed significantly, so that only single times have been selected for each of the series of tests in the DOP range from 1 to 8.

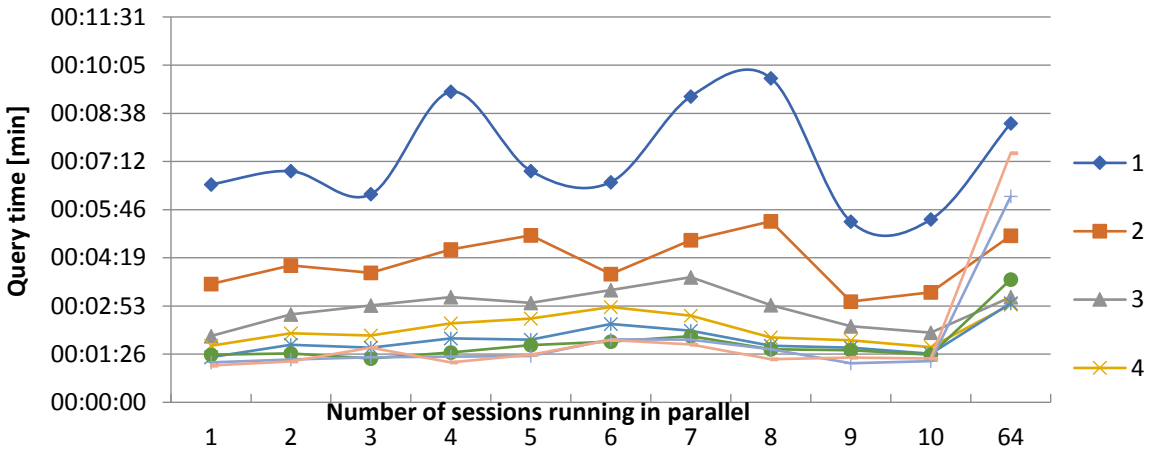

Figure 7. Query times for a full table scan operation launched by multiple sessions simultaneously (CentOS system)

With a view to run several sessions simultaneously, each with a level of parallelism above 1, it was reasonable to expect that the server would start to work much worse. However, as it can be seen in the chart in Figure 7, raising the level of parallelism in the range of 1 to 8 has brought benefits. The curves begin to overlap only from a DOP level of 5, with the number of simultaneous sessions reaching ten. This can be explained by the fact that slow resources of a fast processor can fully cope with the processing of more simultaneous processes. Only the launch of 64 concurrent sessions brought a significant decrease in performance for all DOP levels.

Only in Figure 8 one can observe the problems which appeared while the server was running. Up to the right of the black vertical line, the characteristics of CPU consumption are outlined when performing multisession tests.

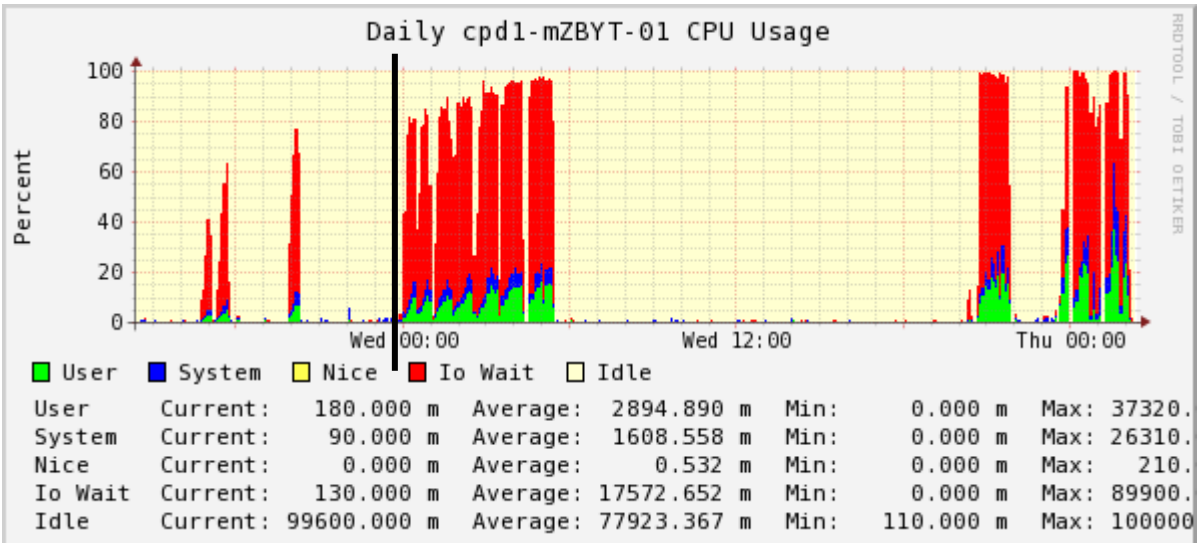

Figure 8. Processor operation characteristics - percentage of consumption (CentOS system)

The 64-parallel session test describes it on the right side of the figure. They show how the server's idle time has decreased. For 64 simultaneous sessions, work on the server was possible, but the slowdown was strongly felt. The waiting time for

any operations in the system shell has increased several times. The attempt to perform the test for 128 simultaneous sessions ended with the server to be blocked and restarted.

Figure 9 shows that only the work of 64 parallel sessions that the HDD was heavily charged. Tests carried out in the range of 1 to 10 sessions showed that disk consumption was about 4,000 operations per second.

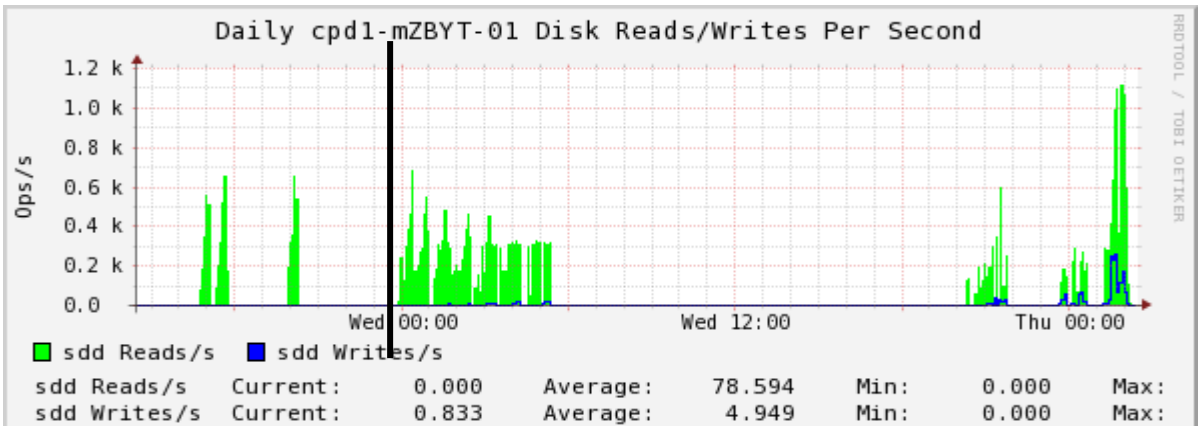

Figure 9. Characteristics of SSD drive - number of reads / writes (CentOS system)

4.4 Test No. 3 - CentOS system

The test has been run once in the most efficient configuration with eight processor cores. The purpose of the test was to check whether the search for a given row of the TEST_TAB1 table could be done faster by using parallel processing or by creating an index on the STR_PAD column and searching with its help.

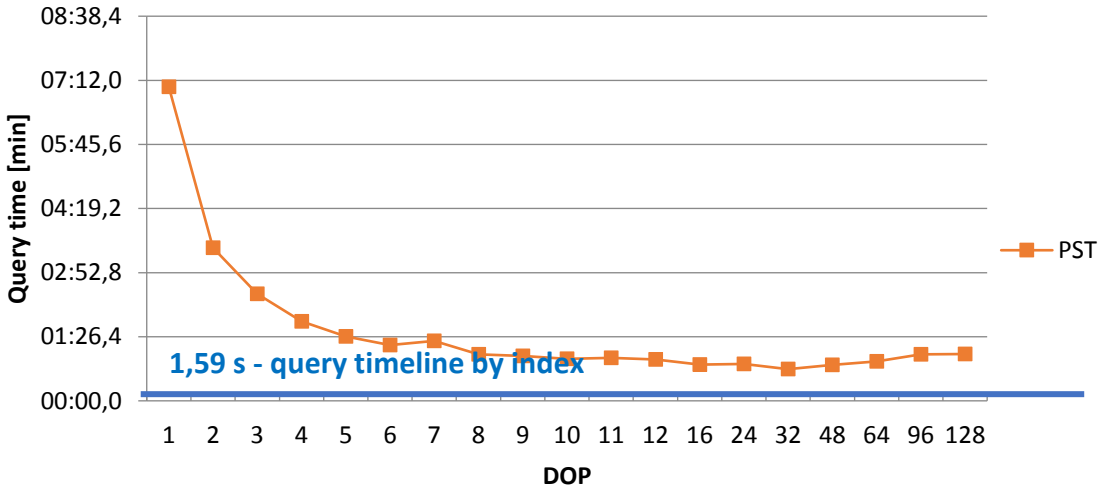

Figure 10. Query times for full table scan operations relative to index time using an index (Windows)

According to Figure 10, the use of an optimal processor configuration for server cannot compete using an efficient index. The best time when using a high DOP level of 32 is 42.7 seconds, which is sometimes almost 27 times slower than searching by index. It should be noted that searching by index does not use parallel processing, which can be seen in Figure 11. Figure 11 presents a plan of the performed query along with the times of individual operations.

Forcing the lack of use of the index (NO_INDEX) caused the optimizer to return to the query plan using parallel processing. As can be seen in Figure 12, there are waiting times for parallel processing operations: PX Deq: Execute Reply, PX Deq: Execution Msg. In addition, using parallel processing, the reading of data from the disk takes place after direct path read, which further introduces a delay in the use of the index.

```
SELECT /*+ parallel(tt1, 2) */ COUNT(*)
FROM test_tab1 tt1
where STR_PAD like 'c%'
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.10       0.10          0       2820          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.10       0.10          0       2820          0           1
Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 66
Number of plan statistics captured: 1
Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         1          1          1  SORT AGGREGATE (cr=2820 pr=0 pw=0 time=101892 us)
    180224     180224     180224   INDEX RANGE SCAN TT1_STRNAME_INDX (cr=2820 pr=0 pw=0 time=2683059 us
cost=2389 size=15390784 card=152384)(object id 84145)
Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                        2        0.00          0.00
  SQL*Net message from client                      2        0.00          0.00
```

Figure 11. Result of trace analysis of operations performed without using the index (CentOS system)

```
SELECT /*+ parallel(tt1, 32) no_index(tt1) */ COUNT(*)
FROM test_tab1 tt1
where STR_PAD like 'c%'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse       33      0.03       0.87          0          0          0           0
Execute     33     25.40    1283.83    1366848    1467738          0           0
Fetch        2      0.20      42.25          0          0          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       68     25.64    1326.97    1366848    1467738          0           1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 66
Number of plan statistics captured: 33

Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         1          0          1  SORT AGGREGATE (cr=2 pr=0 pw=0 time=1294453 us)
        32          1         32   PX COORDINATOR  (cr=2 pr=0 pw=0 time=1294344 us)
         0          0          0    PX SEND QC (RANDOM) :TQ10000 (cr=0 pr=0 pw=0 time=0 us)
         0          1          1     SORT AGGREGATE (cr=44475 pr=41420 pw=0 time=38864164 us)
         0       5461       8422      PX BLOCK ITERATOR (cr=44475 pr=41420 pw=0 time=40240972 us
cost=12911 size=15390784 card=152384)
         0       5461       8422       TABLE ACCESS FULL TEST_TAB1 (cr=44475 pr=41420 pw=0 time=41175427
us cost=12911 size=15390784 card=152384)


Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  os thread startup                                8        0.07          0.27
  PX Deq: Join ACK                                32        0.00          0.00
  PX Deq: Parse Reply                             32        0.05          0.08
  SQL*Net message to client                        2        0.00          0.00
  PX Deq: Execute Reply                          448        0.73         41.90
  PX Deq: Signal ACK RSG                          32        0.00          0.00
  latch free                                       2        0.00          0.00
  PX Deq: Signal ACK EXT                          32        0.01          0.01
  PX Deq: Slave Session Stats                     32        0.00          0.00
  enq: PS - contention                             1        0.04          0.04
  SQL*Net message from client                      2        0.00          0.00
  PX Deq: Execution Msg                          512        3.82         68.04
  direct path read                             21838        0.92       1271.23
  library cache: mutex X                          11        0.01          0.03
  Disk file operations I/O                         8        0.00          0.00
```

Figure 12. Result of trace analysis of operations performed with using the index (CentOS system)

4.5 Test 4 –Windows system

Both the TEST_TAB1 full table scan operation and the hash join of two large TEST_TAB1 and TEST_TAB2 tables were tested for all selected levels of DOP parallelization. The change from test 1 was the use of one of the variants of a fast SSD. The test results are shown in Figure 13.

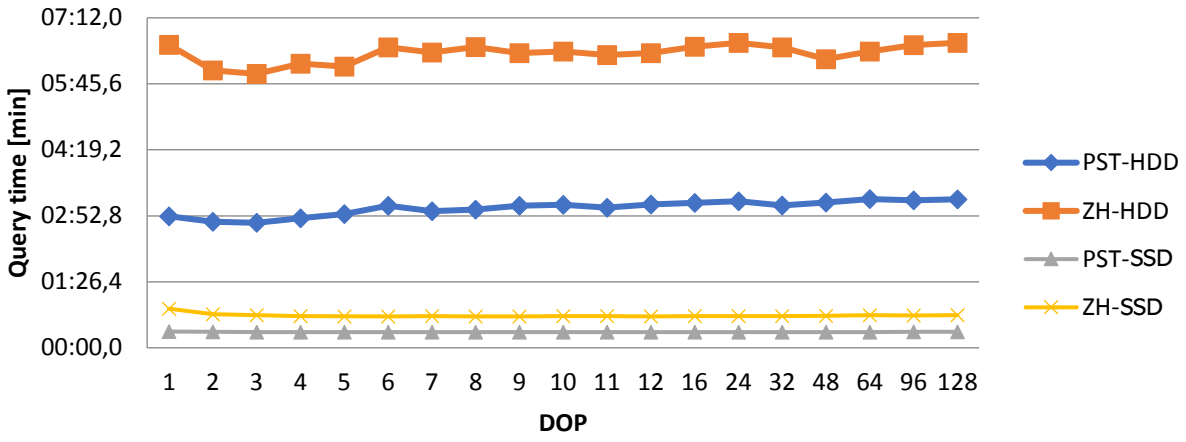

Figure 13. Query times for full table scan operations and hash tables when changing the location of database files from a slower disk to a faster one (Windows system)

Using a faster SSD drive caused that the worst query time at full table scan was better than the lowest time on a powerful Linux server. It is also worth noting that in the case of such large data access times (reading at a level of over 508 MB / s), the role of the processor is limited to a minimum, and thus the level of parallelism is not of practical importance here, which can be seen in the graph above - for full table scanning, the transition from the "weakest" time 21.3 s to the "best" 20.8 s is only 0.5 seconds.

Comparing Figures 14 and 15, we can see that the use of SSD significantly increased the reading of data from the disk in a unit of time compared to the HDD, which RDBMS was able to use and translate into a short time of query execution.

Figure 14. Characteristics of slower HDD - readings per second (Windows system)

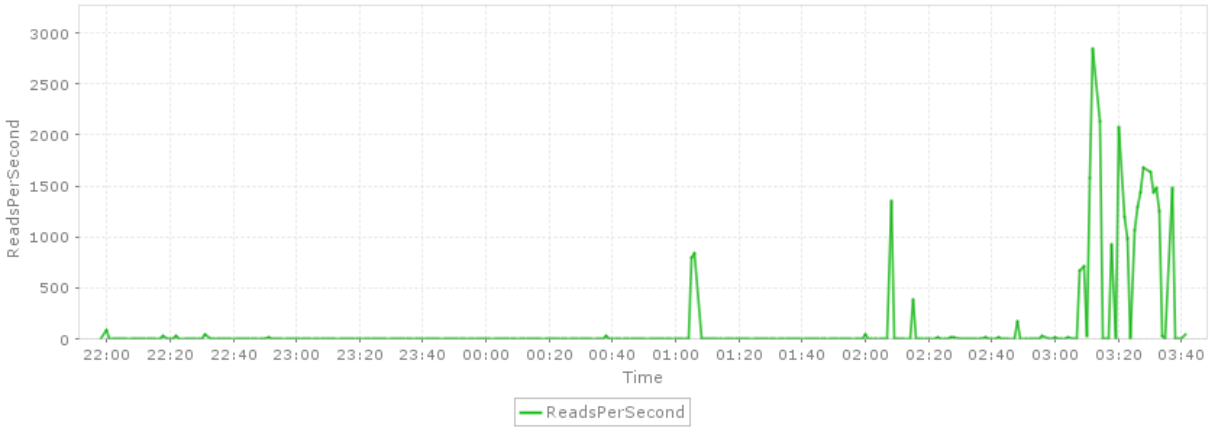

Figure 15. Characteristics of faster disk SSD - readings per second (Windows system)

5. CONCLUSIONS

The results presented above indicate that, under certain conditions, the use of parallel processing can significantly improve the response time of an RDBMS system to an inquiry. The biggest benefit is the introduction of parallel processing itself, i.e. the transition from a DOP level of 1 to 2. However, in the tested cases, the advantage of other hardware and software techniques over the parallel processing technique is clearly visible. Particularly, the use of efficient SSDs, as well as the use of a binary tree index.

Contemporary database systems are forced to deal with an increasing amount of data to be processed. This is solved by investing in newer and more efficient hardware infrastructure. However, it is more important to be able to use your resources using a number of hardware and software solutions to maximize database system performance. This is how to optimize the operation of this system and shorten the time of access to data. The database system consists of a very complex structure.

In the relational system, the data contained in the tables are interrelated, which further hinders the rapid implementation of complex queries. An important issue is therefore the proper database design, optimal configuration of your equipment, fine-tuning of the application code. If the above are complete, one can proceed to the implementation and testing of further methods, such as parallel processing.

The main problems we aimed to address in this paper were: “how to tune the database and the queries addressed to it?”, and “how to choose the DOP parallelization factor so that the use of parallel processing would be profitable”.

Various hardware configurations, application complexity, time required to perform a given operation, number of users - these and many other variables should be taken into account when using parallel processing. First of all, it should be assessed whether its use makes sense.

REFERENCES

[1] ORACLE Database Concepts, https://docs.oracle.com/cd/E11882_01/index.htm.
[2] Kevin Loney, "Oracle Database 11g The Complete Reference", ISBN-10: 0071598758.
[3] ORACLE Database Performance Tuning Guide, https://docs.oracle.com/cd/E11882_01/server.112/e41573/toc.htm.
[4] ORACLE Database Reference, PARALLEL ADAPTIVE MULTI USER, https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/PARALLEL_ADAPTIVE_MULTI_USER.html#GUID-6A4F12CA-E6EA-4D13-A725-80B86404ECFA.

[5] Parallel Execution with Oracle Database, WHITE PAPER/FEBRUARY 20, 2019, https://www.oracle.com/technetwork/database/bi-datawarehousing/twp-parallel-execution-fundamentals-133639.pdf
[6] Douglas Ian Burns, "How Many Slaves?", http://oracledoug.com/serendipity/index.php?/archives/840-How-Many-Slaves.html.
[7] SQL trace, 10046, trcsess and tkprof in Oracle, https://oracle-base.com/articles/misc/sql-trace-10046-trcsess-and-tkprof.
[8] Wajszczyk Bronisław Ignacy, Biernacki Konrad : Optimization of the efficiency of search operations in the relational database of radio electronic systems, Proceedings Volume 10715, 2017 Radioelectronic Systems Conference; 107150H (2018).
[9] Matuszewski J., Paradowski L., "The Knowledge Based Approach for Emitter Identification", 12th International Conference on Microwaves and Radar (MIKON). Krakow, Poland, May 20-22, 1998, Vol.3, pp. 810-814, DOI: 10.1109/MIKON.1998.742832 (1998).
[10] Matuszewski J.: The Radar Signature in Recognition System Database. 19th International Conference on Microwaves, Radar and Wireless Communications MIKON-2012, Warsaw, art. no. 6233565, pp. 617-622, DOI: 10.1109/MIKON.2012.6233565. (2012)
[11] Li, J. & Zhang, "Cluster based parallel database management system for data intensive computing", W. Front. Comput. Sci. China 3: 302. https://doi.org/10.1007/s11704-009-0031-5. (2009).
[12] Pushpa Rani Suri, Sudesh Rani, "A New Classification for Architecture of Parallel Databases", *Information Technology Journal*, vol. 7, pp. 983. (2008).
[13] Todd Eavis, Ahmad Taleb, "Query Optimization and Execution in a Parallel Analytics DBMS", *Parallel & Distributed Processing Symposium (IPDPS) 2012 IEEE 26th International*, pp. 897-908, (2012).
[14] N.TomovE, DempsterM.H, Williams.A, BurgerH., Taylor P.J.B., King. P, Broughton, "Analytical response time estimation in parallel relational database systems", School of Mathematical and Computer Sciences, Heriot-Watt University, Riccarton, Edinburgh EH14 4AS, UK. https://doi.org/10.1016/j.parco.2003.11.003
[15] C. S. Pan, M. L. Zymbler, "Development of a parallel Database Management System on the Basis of Open-Source POSTGRESQL DBMS", Vestnik YuUrGU. Ser. Mat. Model. Progr., 2012, no. 12, 112–120.
[16] Pietkiewicz, T., Wajszczyk, B., "Fusion of identification information from ELINT-ESM sensors", Proceedings of SPIE 10715, 2017 Radioelectronic Systems Conference, 107150F (2018).
[17] Pietkiewicz, T.," Removal of conflicts in fusion of identification information from ELINT-ESM sensors", Proceedings of SPIE 11055, XII Conference on Reconnaissance and Electronic Warfare Systems, 110550S (2019).