ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20200914224607
Profesional paper

A Comparison of Query Execution Speeds for Large Amounts of Data Using Various DBMS Engines Executing on Selected RAM and CPU Configurations

Slaviša ILIĆ, Siniša ILIĆ, Ivan MILOVANOVIĆ, Petar SPALEVIĆ*, Dragiša MILJKOVIĆ

Abstract: In modern economies, most important business decisions are based on detailed analysis of available data. In order to obtain a rapid response from analytical tools, data should be pre-aggregated over dimensions that are of most interest to each business. Sometimes however, important decisions may require analysis of business data over seemingly less important dimensions which have not been pre-aggregated during the ETL process. On these occasions, the ad-hoc "online" aggregation is performed whose execution time is dependent on the overall DBMS performance. This paper describes how the performance of several commercial and non-commercial DBMSs was tested by running queries designed for data analysis using "ad-hoc" aggregations over large volumes of data. Each DBMS was installed on a separate virtual machine and was run on several computers, and two amounts of RAM memory were allocated for each test. Measurements of query execution times were recorded which demonstrated that, as expected, column-oriented databases out-performed classical row-oriented database systems.

Keywords: data volume performance impact; hardware contribution to DBMS performance; on-line data aggregation; row-versus column oriented DBMS

1 INTRODUCTION

Online Transactional Processing (OLTP) DBMSs are designed to provide rapid concurrent access for a large number of users, allowing them to record and process large volumes of data over time. However, as a consequence of providing this flexibility, these systems need a lot of time to perform the complex queries and generate summary reports required by management. In contrast, systems oriented towards ad-hoc queries of large amounts of data are optimised for fast data retrieval, tables de-normalised and their data made redundant, in order to accelerate searches and queries.

Most of today's Database Management Systems (DBMS) implement row-oriented data storage. With this architecture of data storage, one writing cycle is sufficient to store all the data in a given row on the disk, thus achieving high writing speed. The other approach is column-oriented data storage, which is ideal for read-intensive workloads because it can significantly reduce disk I/O compared to the more-typical row-based storage. This is because, to process a query, a row store must read all columns in all of the tables named in the query, regardless of how wide the tables might be or how many columns are actually needed. A column store with a query-specific projection reads only a subset of columns. Thus architectures with the columnar storage in large data sets should provide much better performance.

Data in OLAP databases are usually collected from multiple OLTP sources using ETL tools. Extract, Transform and Load (ETL) is an important process that comprises the extraction of data from various data sources, the transformation of extracted data according to business requirements, and loading that data into a data warehouse. One of the transformation processes needed to prepare data to assist business decisions is aggregation, but generally not all of the available "fact" data is aggregated in an ETL process. Near real time reporting, statistics and aggregation became the norm, requiring the expeditious processing of data in order to rapidly generate reports. Businesses need to respond promptly to opportunities, and they must make business decisions more quickly than their competition [1]. It is important that a database system used for analytical purposes should be able to adequately process large amounts of non-pre-aggregated data, and that the data processing is performed on-line. This paper focuses on this performance aspect, as this is usually neglected in similar database performance comparisons.

As of July 2019, ranking of relational DBMS systems by popularity, is as follows [2]: 1st Oracle, 2nd MySQL, 3rd MS SQL Server, 8th MariaDB, and 17th HP Vertica

The aim of this paper is to compare the on-line processing performance of some of the most popular commercial and open source DBMSs (both column-oriented and row-oriented systems) intended for the analysis of large amounts of data. The following commercial DBMS platforms were selected due to their availability to download and install: HP Vertica Community Edition (CE) (with no time limits or license requirements), Microsoft SQL Server (with Microsoft Imagine licence) and Oracle (with a Developer License), and some popular open source DBMS: MySQL with InfoBright Engine and MariaDB ColumnStore.

The paper contributes the following: An investigation of the speed of "online" aggregation of large volume of data between various types of DBMS (row-oriented and column oriented, commercial and open-source) executing on a range of CPUs (with different speeds) and memory RAM is performed; we show that the better hardware (faster CPU speed and amount of RAM) creates the faster query response; and we prove that query execution speed of open source column-oriented DBMS is of the same order of magnitude compared to the performance of the leading commercial DBMSs. We stress that every performance result is obtained in extensive experiments with a number of repetitions in order to eliminate stochastic effects. The result of our work might be of interest to potential clients who need data analysis services.

2 RELATED WORK

Many papers have been published with analyses and comparisons of modern DBMS. To test the advantages and disadvantages of various DBMS, different features and architectures are selected: Open Source, Commercial, Centralised, Distributed, on Cloud, In-Memory, NoSQL, on Virtual Machines etc.

There are several standard benchmarks available as decision support tools [3]. They provide results for the performance status of the OLTP database (TPC-H), examine large volumes of data including the complete flow from the sources to the target data warehouse (TPC-DS) and test DBMS running in a virtualisation environment (TPC-VMS). But, in order to obtain valid results, certain conditions must be fulfilled related to database availability, size of database, number of tuples in tables, etc. Comparisons of the performances of DBMSs need to describe the methodology adopted, with defined indicators and standardised evaluation criteria. The methodologies are discussed in several papers.

Osman et al. [4] present a categorization of queueing network performance models of database systems and a survey and evaluation of performance evaluation methodologies proposed to map database system specifications onto queueing network models with appropriate workloads. Seng et al. [5] state that test results from standard benchmarks such as the TP1, TPC-A, TPC-B, TPC-C, TPC-D, TPC-H, TPC-R, TPC-W, Wisconsin, and AS3AP are estimates of possible system performance for certain pre-determined application types and that DBMS performance on an actual database domain may vary significantly from that predicted by the standard benchmarks. Today's business applications are no longer the old-styled legacy batch applications; rather most data processing activities occur on-line. Therefore, good performance is essential to these Distributed Real-Time DBMS applications, as discussed in [6].

Some authors have performed reviews of the performance of some DBMSs based on their characteristics derived from the literature, while others performed measurements on specific DBMSs using proprietary methods.

A comparison of the essential characteristics (including architecture, implementation technology and function features) of several recent open-source cloud storage platforms, such as Hadoop, abiCloud, Cassandra and MongoDB, is presented in [7]. A benchmark analysis based on a version of the standard TPC-W is described in [8], based on a dataset with 288,000 tuples using DB partitions with data distributed across several cloud providers and their different programming interfaces. In the same paper, results are presented of the same benchmark that is executed using MySQL and Oracle Express RDBMS.

The performance of the popular open source DBMSs, MariaDB and MySQL, using the processor and memory resources of Virtual Manager Virtual Machine Xen 4.4, is presented in [9]. The measurements of query execution response time between "conventional" and dedicated database machine architectures using various types of communication channels are performed by Yao et al. in [10].

Comments on the various NoSQL systems and a comparison (using multiple criteria) between them of some features (such as persistence, replication, high availability, transactions, etc.) and performance (read/write latency) are presented in [11], and another review of the NoSQL and RDBMS concepts for handling big data, requirements, architecture and features is presented in [12]. The performance of commercial (MS SQL Server, Oracle, DB2, HP Vertica) and open source DBMSs (MySQL, Cassandra, Apache Hive) during import and export of data, performing DML (Data Manipulation Language) operations and querying using different approaches are compared in [13-18].

3 DBMS PLATFORMS USED IN EXPERIMENTS

In order to achieve uniformity and execute a fair comparison of query speeds of different DBMSs, the following DBMSs were installed on separate virtual machines (VM) running on VMware Workstation Player: HP Vertica, Microsoft SQL Server, Oracle, MariaDB ColumnStore and MySQL. They were configured to have the same hardware settings: the same amount of (DDR3) RAM (with two different settings of 4 GB and 8 GB for the amount of RAM assigned to the guest VMs) and a quad-core processor each. All virtual machines were stored on the computers' internal SATA HDDs. The host OS for all VMs was Windows 10 Professional with 16 GB of RAM memory.

Two sets of experiments were performed in two different (older and newer) environments. In the first experiment (with an older OS and DB), the installation comprised the following versions of the databases and operating systems: 1) *HP Vertica Analytic Database CE 6.0.0* RDBMS on CentOS 5.5 x86_64, 2) *MySQL 5.6* RDBMS with two separate instances: one with InnoDB and another with InfoBright storage engine on CentOS 6.3 x86_64, 3) *Oracle 11g R2* RDBMS with Oracle's Data Warehouse template, on CentOS 6.5 x86_64 and 4) *MS SQL Server 2014 Enterprise Edition* on MS Windows Server 2008 SP1 x64.

In the second experiment (with a more recent OS and DB), the installation comprised the following versions of the databases and operating systems: 1) *HP Vertica Analytic Database 9.0.1 CE* pre-installed on CentOS 7 x86_64 (CentOS7_64), 2) *MariaDB ColumnStore* database version 10.2.13 installed on CentOS7_64, 3) *Oracle DB Server 12c* release 12.1.0 Enterprise installed on CentOS7_64 and 4) *MS SQL 2017 Server (ver. 14.0) Standard Edition* installed on MS Windows Server 2016.

The tests were performed on three different computers: (1) Toshiba Qosmio X775 with I7-2630QM processor running at 2 GHz, (2) HP Envy 17-r104nn with I7 6700HQ processor running at 2.6 GHz, and (3) an unbranded desktop computer with I7-2600K processor running at 3.4 GHz clock speed.

In some of the "older" DBMSs (MySQL 5.6 with InnoDB, Oracle 11g R2, and MS SQL Server 2014 (DataBase 1 using RowStore), standard indexes on foreign keys of "fact" tables are created. In HP Vertica 6.0 DBMS, only foreign keys constraints on "fact" tables are created. In the DBMS MS SQL Server 2014 (DataBase 2 using ColumnStore), non-clustered ColumnStore indexes on complete tables are created, and for MySQL using InfoBright engine, indexes are not created.

In the "newer" DBMS, Oracle 12c, HP Vertica 9.0 and MariaDB, ColumnStore indexes are not created. In MS SQL Server 2017, non-clustered ColumnStore indexes on complete tables are created.

The configuration files (my.cnf in MySQL and in MariaDB, my-ib.cnf in MariaDB) and settings (spfile in Oracle) are set according to the amount of RAM assigned to the guest VM.

3.1 Architectures of DBMS Used

HP Vertica was explicitly designed for analytic workloads rather than for transactional workloads. HP Vertica's free versions (HP Vertica Community Edition) are the Enterprise Edition (ver. 6.0) and Analytics Platform (currently ver. 9.1) available for download in the form of a virtual machine (https://www.vertica.com/log-in/) on which operating system CentOS (64bit version) is preinstalled.

The Vertica DB physically organises table data into projections, which are sorted subsets of the attributes of a table. In practice, most applications have one super projection (containing every column of the anchoring table) and zero to three narrow, non-super projections [19]. Each projection has a specific sort order on which the data is totally sorted. HP Vertica uses Read Optimized Store (ROS) and Write Optimized Store (WOS). Data in the ROS is physically stored in multiple ROS containers on a standard file system. Each ROS container logically contains some number of complete tuples sorted by the projection's sort order, stored as a pair of files per column. HP Vertica is a true column store - column files may be independently retrieved as the storage is physically separate. By dividing each on-disk structure into logical regions at runtime and processing the regions in parallel, HP Vertica achieves high performance.

Oracle 11g R2 database can be installed using one of several templates: Transaction Processing, General Purpose, Data Warehouse and Custom Database, through its built-in tool Database Configuration Assistant (DBCA). The main characteristic of the Data Warehouse template used in this experiment is that it sets the *star_transformation_enabled* system parameter to a value of "true", and it is used when users perform numerous, complex queries that process large volumes of data. Star transformation optimizer [20] was introduced in Oracle 8i to process star queries efficiently. Oracle 11g R2 Data Warehouse template enables In-Memory parallel execution that takes advantage of a large aggregated database buffer cache [21].

The **Oracle 12c** development team realised the goal of combining several technologies in one single product - "Traditional" DB storing data on a disk as rows, and "analytical" In-Memory storing data in a columnar format - features which so far were only available in separate products [22]. IM Column Store enables a customer to have both OLTP and Data Decision Support systems in one product. Data are persistently stored on disk, and they are stored in row format. When data are requested for read/write operations they are loaded into the traditional Row Store (Buffer Cache), and when data are requested for read-only operations they are populated into the In-Memory Column Store. This includes a transformation from row to columnar format; whenever a transaction that includes inserts, up-dates, or deletes is committed, the new data will appear in both formats. The IM column store environment in Oracle 12c enables a user to choose to store specific groups of columns, whole tables, materialised views or table partitions in the store [23]. Thus the IM column store is good for queries that aggregate data.

MS SQL Server column store indexes are "pure" column stores, not a hybrid, because they store all data for each column on separate pages. This improves the I/O scan performance and makes more efficient use of memory [24]. A column store index stores its data column-wise in compressed form and is designed for fast scans of complete columns. Each column segment is stored as a separate BLOB (Binary Large OBject). Query speed is increased also by using optimization for the latest generation of CPUs, batch hash joins and new methods for handling multiple joins (by collapsing inner joins into a single n-ary join operator). Microsoft SQL Server Column Store (CS) index [25] is designed to work well with data warehouses, especially for storing large data 'fact' tables. These indexes offer excellent performance at full table scans, and they are not designed for searching specific values. Two types of indexes in MS SQL CS can be implemented both clustered and non-clustered. The clustered CS index cannot be combined with other indexes; it is the primary storage for the whole table. A non-clustered index is a secondary index, it can index a subset of columns in the clustered index or heap and it can be combined with the other indexes on the table.

MariaDB ColumnStore (CS) is a columnar storage engine that utilizes a massively parallel distributed data architecture. It is a columnar storage system built by porting InfiniDB 4.6.7 to MariaDB 10.1. Storing table data in columns rather than rows allows the query optimizer to only read columns necessary to fulfil a given query and its result set. By storing data as columns it is also much easier to add and remove columns over time even online. MariaDB CS is designed for big data scaling to process petabytes of data, linear scalability and exceptional performance with real-time response to analytics queries. It leverages the I/O benefits of columnar storage, compression, just-in-time projection, and horizontal and vertical partitioning to deliver tremendous performance when analysing large data sets [26].

The **MySQL** database is primarily specialised for OLTP transactions. However, MySQL can be used even for Enterprise Data Warehousing. From the MySQL storage engine reference guide for data warehousing [27], based on comparative characteristics of the offered engines, the Infobright engine was found to be the best. The Infobright engine features are the following: lowest storage cost, knowledge grid indexes, fastest data loads, column-oriented tables and data compression, and it is the best for Traditional or Historical DW. The Infobright storage engine is a column-oriented architecture that combines a high-speed data loader, strong levels of data compression, a smart, external optimizer and "knowledge grid" to deliver impressive data warehousing capabilities. It serves as a substitute for indexes, which means that indexes are not needed on an Infobright table [28].

4 TEST PREPARATION

All tests on the DBMSs were performed using the same set of tables and content. The built-in Vmart Example Database in HP Vertica's VM, which is designed to test HP

Vertica's features and performances, was chosen. The initial vmart_define_schema.sql script can be found in the pre-configured HP Vertica VM archive (in /opt/vertica/examples/VMart Schema/) but, as a result of different "dialects" of the DBMSs being tested, the SQL syntax was adjusted for each specific database. The Vmart database consists of three snowflake schemas: **Public** schema, **Store** schema and **Online_Sales** schema.

Four "fact" tables were important to the experiment due to the number of rows in those tables: **inventory_fact** and **store_orders_fact** with 300000 rows, and **store_sales_fact** and **online_sales_fact** with 5 000 000 rows. All tables from the Vmart Example Database (that were installed on the HP Vertica) were exported in .csv format. The database was installed on each DBMS, by firstly running the scripts to create the schemas, then loading the data from the .csv files using the batch loading tools available for each database. The load and export performances of each DBMS used have been measured and compared [29].

Four different query tools were used to connect to the databases and run the SQL queries. All tools were installed on the VMs along with databases. RazorSQL was used to connect to HP Vertica 6.0, Oracle SQL Developer to connect to Oracle DBMS, MySQL Workbench to connect to MySQL and MariaDB ColumnStore, and SQL Server Manage-ment Studio to connect to MS SQL Server. The first three tools, RazorSQL, SQL Developer and MySQL Workbench, have pre-built functionality that shows query execution time. In SQL Manage-ment Studio, the "Include Client Statistics" option was activated to record the "total execution time" along with other relevant statistical parameters of query execution.

Four queries were created that were designed to measure "ad-hoc" DBMS query retrieval performance. The goal of the measurements was to load and process all rows of "fact" tables rather than to implement filters (using WHERE clauses) which would have reduced the number of loaded and processed rows. All queries were written using PL/SQL syntax, minor modifications were made for the different SQL "dialects".

Table 1 The list of queries executed in the experiments

Query 1:
```
SELECTprom.price_reduction_type, prod.category_description,
  sum(o.gross_profit_dollar_amount) AS sum_profit
FROMonline_sales.online_sales_fact o INNER JOIN public.promotion_dimension prom ON
  o.promotion_key = prom.promotion_key INNER JOIN public.product_dimension prod ON
  o.product_key = prod.product_key AND o.product_version = prod.product_version
GROUP BYprom.price_reduction_type, prod.category_description;
```

Query 2:
```
SELECTw.warehouse_region, p.category_description, sum(i.qty_in_stock) as sum_qty
FROMpublic.inventory_facti INNER JOIN public.warehouse_dimension w ON
  i.warehouse_key = w.warehouse_key INNER JOIN public.product_dimension p ON
i.product_key = p.product_key AND i.product_version = p.product_version
GROUP BYw.warehouse_region, p.category_description;
```

Query 3:
```
SELECTd.calendar_year, ROUND((SUM(CASE WHEN cd.customer_region = sd.store_region
 THEN 1 ELSE 0 END) / COUNT(*)) * 100, 2) AS percentage_same_region,
 SUM(gross_profit_dollar_amount) AS sum_profit
FROMstore.store_sales_fact sf
  INNER JOIN public.date_dimension d ON sf.date_key = d.date_key
  INNER JOIN store.store_dimensionsd ON sf.store_key = sd.store_key
  INNER JOIN public.customer_dimension cd ON sf.customer_key = cd.customer_key
GROUP BYd.calendar_year;
```

Query 4:
```
SELECTshipper_name, EXTRACT(year from date_ordered) AS calendar_year,
  ROUND((SUM(CASE WHEN date_delivered>expected_delivery_date THEN 1 ELSE 0 END)
  /COUNT(*))*100, 2) AS percentage_latency,
  ROUND(AVG(DATEDIFF(date_delivered, expected_delivery_date)), 2) AS delivering_days,
  ROUND(AVG(DATEDIFF(date_ordered, date_shipped)), 2) AS shipping_days
FROMstore.store_orders_fact
GROUP BYshipper_name, calendar_year;
```

The goal of **Query 1** was to measure the performance of aggregation in one "fact" table of 5 million rows, inner joined with one column of the "dimension" table **promotion_dimension** of 1000 rows, and on two columns of the "dimension" table **product_dimension** of 60 000 rows. The aggregation was calculated on currency and the query grouped by two columns belonging to the "dimension" tables. The query calculated the profit achieved, grouped by the categories of products and the types of price reductions.

The goal of **Query 2** was to measure the performance of aggregation in one "fact" table of 300 000 rows, inner joined with one column of the "dimension" tables **warehouse_dimension** of 100 rows, and on two columns of the "dimension" table **product_dimension** of 60 000 rows. The aggregation was calculated on quantity and the query grouped by two columns that belong to "dimension" tables. This query showed the number of products that are stored in warehouses, grouped by the regions and the categories of product.

The goal of **Query 3** was to measure the performance of aggregation in one "fact" table of 5 million rows, inner joined with three "dimension" tables (**date_dimension** of 1826 rows, **store_dimension** of 250 rows, and

customer_dimension of 50 000 rows) on one column of each table. The query comprised two aggregation functions and one CASE clause within the aggregation function. In addition, the ROUND function was implemented on the aggregation function. This query calculated the profit of the Vmart supermarket chain and percentage of realised profit from regional customers grouped by calendar years.

The goal of **Query 4** was to measure the performance of aggregation in one "fact" table of 300 000 rows without any joins to "dimension" tables, but with multiple date and mathematical functions (EXTRACT, DATEDIFF, ROUND, SUM, AVG). This query calculated the average amount of time (expressed in days) shippers required to deliver goods to Vmart, and the percentage of delivery latency; the results were grouped by the shipper names and calendar year of delivery.

5 RESULTS AND DISCUSSION

In order to obtain fair results, one query is executed per VM boot, thus avoiding the use of cache data (containing pre-loaded tables) when executing the following query. Cached data impacted the following query execution despite restarting the DB service - the speed of the same query execution was always faster after restarting the DB service compared to the execution time after restarting the whole VM.

Queries were executed following the reduction of the system load to below 0.2 after booting the VMs. The measurements were repeated three times per query, respecting the one query per boot rule, and the results averaged for presentation. In order to perform "fair" comparisons of the "ad-hoc" queries, as if they were being executed for the first time, the query statistics created and stored after each query by MS SQL to accelerate subsequent queries, were deleted.

The results obtained for the execution speeds of queries Q1 and Q3 of MySQL with the InnoDB engine and MS SQL Server with standard RowStore indexes on the slowest computer were excessively slow, so queries on these databases using the other configurations were not run.

The results of the query execution speeds of Experiment 1 and Experiment 2 are shown in Tabs. 2 and 3 respectively. In both tables, the symbols "+" and "−" indicate "exceptions" where the same query executed faster using less memory (4 GB) on the same computer, and the symbol "*" denotes "exception" that the same query executed faster on the "older" version of the DBMS using the same computer.

Table 2 Queries execution time (in ms) for "older" DBMSs on different computers and different memory assigned in Experiment 1

Processor / RAM	Queries	MS SQL 2008 (columnstore)		HP Vertica 6.0		MySQL 5.6 (IB)		Oracle 11g datawarehouse		MS SQL (rowstore)	MySQL (InnoDB)
		4 GB	8GB	4 GB	8 GB	4 GB	8 GB	4 GB	8 GB	4 GB	4 GB
I7-2630QM 2.0 GHz DDR3 RAM 1333 MHz CL9	*Q*1	3182	3088	4423	2750	6023	5018	9531	8026	79633	214399
	*Q*2	2090	2037	456	272	1018	1015	1699	1056	710	6433
	*Q*3	3416	3244	3899	3329	6012^{-}	6019^{+}	9526	9522	12320	87719
	*Q*4	1014	858	318	318	1010^{-}	1012^{+}	1009^{-}	1012^{+}	103	1321
I7- 6700HQ 2.6 GHz DDR3 RAM 1600 MHz CL11	*Q*1	2303^{-}	2414^{+}	2873	1991	5151	4527	7037	6520		
	*Q*2	2086	2187	550	432	508	508	507^{-}	509^{+}		
	*Q*3	2144	2109	2210*	1575	5013^{-}	5030^{+}	8022	7600		
	*Q*4	952	1123	303	280	1008	1007	1009^{-}	1011^{+}		
I7-2600K 3.4 GHz DDR3 RAM 1600 MHz CL11	*Q*1	1981	1775	2463	1539	4017^{-}	4021^{+}	5016	5015		
	*Q*2	998	951	415	376	571	543	506	506		
	*Q*3	1887	1703	1579	1075	4666	4513	5521	5514		
	*Q*4	530	530	203^{-}	207^{+}	1007	507	1011	1011		

Table 3 Queries execution time (in ms) for "newer" DBMSs on different computers and different memory assigned in Experiment 2

Processor / RAM	Queries	MS SQL 2017 CS		HP Vertica 9.0.1		MariaDB CS 10.2.13		Oracle 12c In-Memory CS	
		4 GB	8 GB	4 GB	8 GB	4 GB	8 GB	4 GB	8 GB
I7-2630QM 2.0 GHz DDR3 RAM 1333 MHz CL9	*Q*1	3358	2414	3490	1720	3936	3779	3272	2915
	*Q*2	844	836	357	323	1305	1262	498	382
	*Q*3	2594	2875	2075	1059	4452	4282	4312	4148
	*Q*4	875	795	315	311	1042	1022	922	610
I7- 6700HQ 2.6 GHz DDR3 RAM 1600 MHz CL11	*Q*1	1575	1515	2769	1483	3124	2927	2289	2193
	*Q*2	907	440	413	282	542	489	473	252
	*Q*3	2008	1915	2265*	936	3889	3285	3844	3671
	*Q*4	718^{-}	796^{+}	200^{-}	222^{+}	732	716	646	584
I7-2600K 3.4 GHz DDR3 RAM 1600 MHz CL11	*Q*1	1516	1500	2498	1371	2741^{-}	2745^{+}	2057	1928
	*Q*2	392	461	236	214	466^{-}	483^{+}	253	222
	*Q*3	1606	1579	1482	803	2347	2308	2930	2778
	*Q*4	516	497	292	163	344^{-}	404^{+}	433	363

As was to be expected, most queries executed faster under the following conditions: more powerful computers, with increased RAM, and on "newer" versions of DBMS. The execution times of queries *Q*1 and *Q*3 with "fact" tables of 5M rows were quickest on all computers using HP Vertica 9.0 with 8 GB of RAM. When 4 GB was allocated to the VM, query *Q*1 executed faster on MS SQL 2017 whilst *Q*3 executed faster on HP Vertica 9.0. Both queries executed the slowest using Oracle 11g; that was expected as it does not implement CS indexes. Despite slower execution times compared to DBMSs that support CS indexes, the execution speed of Oracle 11g was significantly better than the execution times of the same queries in MS SQL 2014 and MySQL (InnoDB), both of which implement the RowStore format instead of CS indexes.

Fig. 1 presents a comparison of the execution times of query *Q*1 across architecture, amount of RAM allocated to the VM, and DBMS. The execution time is in line with the performance of the computers' architectures, and generally query Q1 executes faster in VMs with the larger RAM. The DBMSs are sorted (left to right) according to the execution times on each computer, split into two groups by VM RAM.

MS SQL 2017 executed Query Q1 faster than Q3 in all cases (except on the computer with I7-2630QM processor with 4 GB assigned to the VM), Oracle 12C in all cases, Oracle 11g R2 in all cases (except on the slowest computer with 4 GB assigned to the VM), MariaDB ColumnStore on the slower computers, and MySQL InfoBright on the fastest computer and on the slower computers with 8 GB allocated for the VM. In other cases, query *Q*3 was executed faster. A similar situation arose with queries *Q*3 and *Q*4. In some cases, *Q*2 was faster (in Oracle 12c on all computers, on faster computers in MySQL IB and Oracle 11G, etc.) and in some other *Q*4 was faster. The last two queries (*Q*2 and *Q*4) ran at comparable speeds on DBMSs with either type of index as a result of the relatively low number of rows ("only" 300 000) in the "fact" table.

Better computers would be used in a production environment, so the execution times of queries Q1 and Q3 were taken using a "fact" table of 5 million rows on the fastest computer and the results are presented in Fig. 2.

In Fig. 2 the DBMSs are sorted (left to right) according to the execution time of query *Q*3. From the figure it can be seen that the execution times of queries *Q*1 and *Q*3 are under 5.6 seconds in all DBMSs on the fastest computer. The best execution times are recorded for HP Vertica and MS SQL. From Fig. 3 it can be seen that queries *Q*2 and *Q*4 executed fastest using HP Vertica, followed by Oracle 12C and MariaDB ColumnStore. The best execution time with HP Vertica for those queries was about 200 ms.

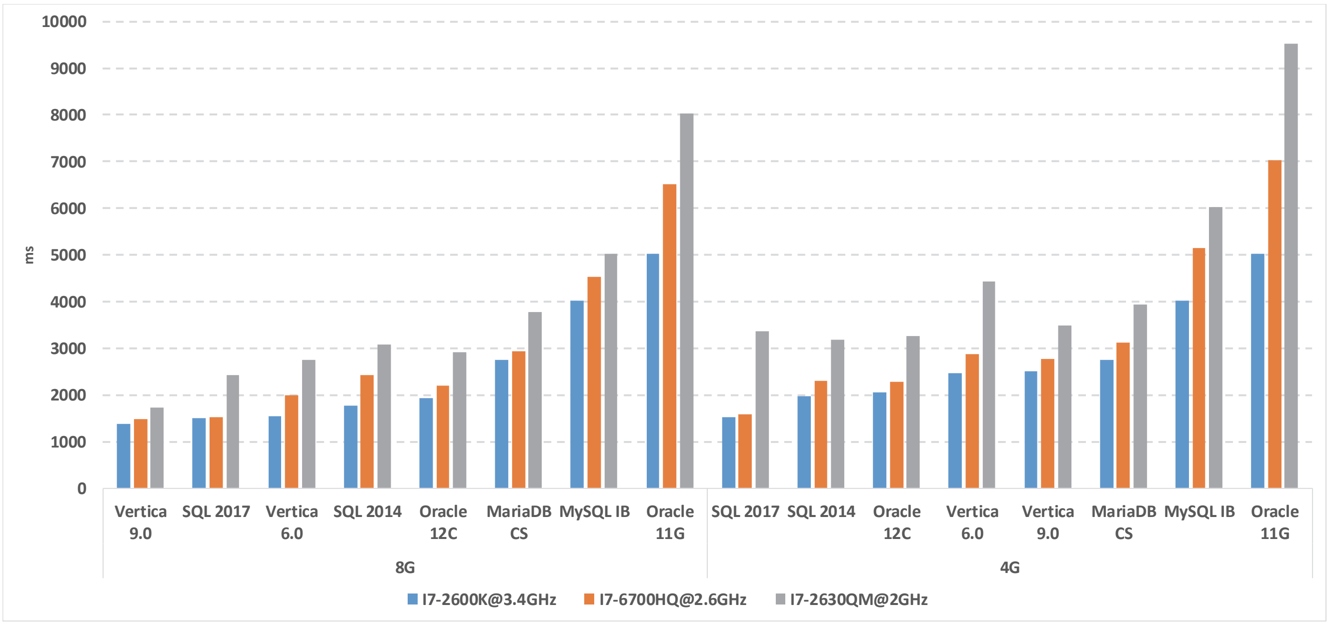

Figure 1 Execution speed of query Q1

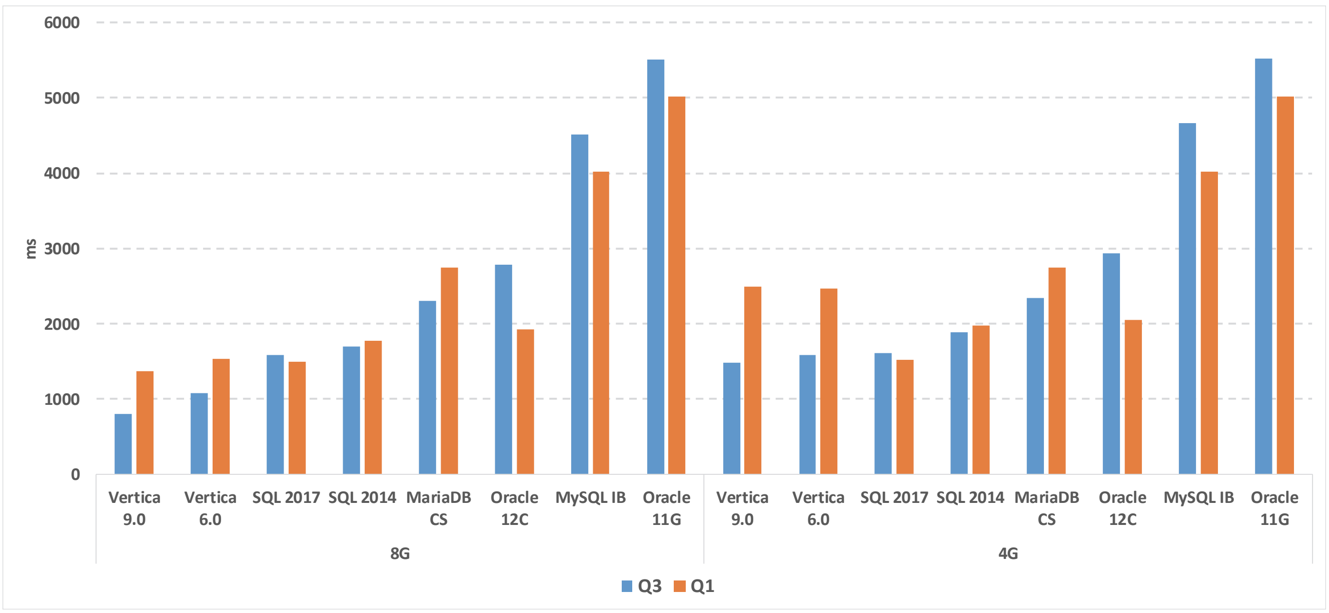

Figure 2 Execution speed of queries Q1 and Q3 with larger fact tables on fastest computer used in experiments

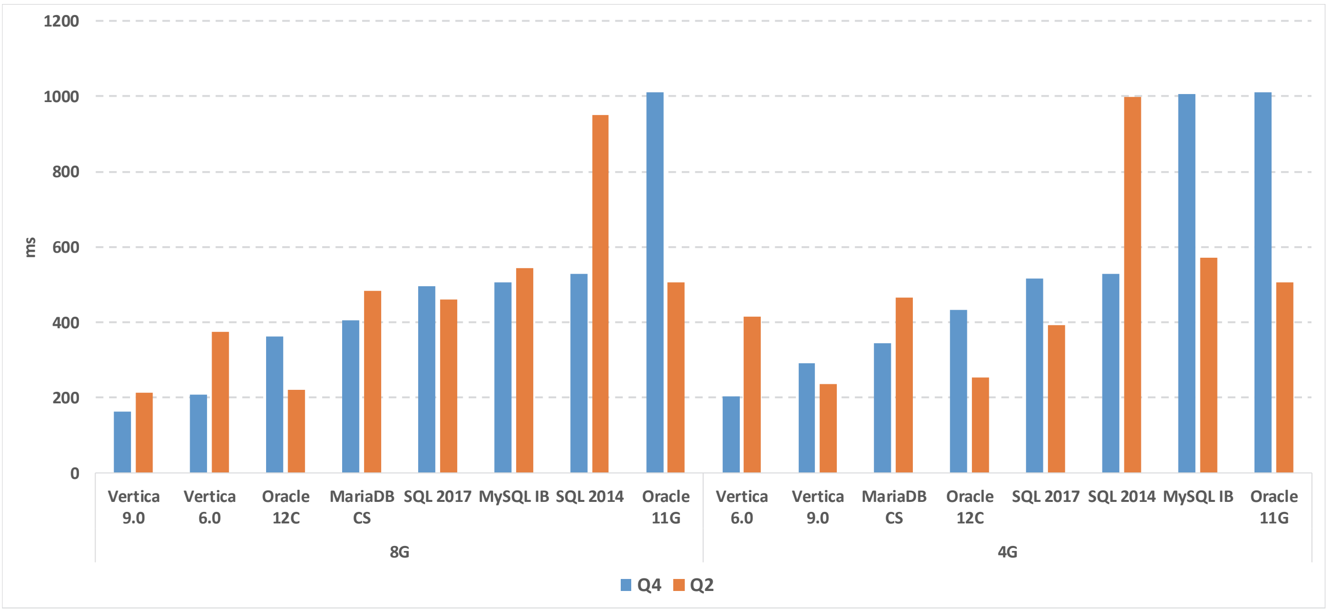

Figure 3 Execution speed of queries Q2 and Q4 with smaller fact tables on fastest computer used in experiments

In general, the greater the number of rows in a table, the slower the query execution speed, as was to be expected. Better computer performance mostly led to faster query execution, but increasing the RAM assigned to the VM did not always contribute to an increase in query execution speed. Queries in databases with ColumnStore indexes implemented, predictably executed significantly faster.

6 CONCLUSION

This paper presents a comparison of the query performance of some of the modern database systems on large volumes of non-pre-aggregated data. The general assumption that DBMS with column-oriented storages will out-perform the classic row-oriented type is proved. All the tested DBMS (MS SQL Server, HP Vertica, Oracle, MariaDB ColumnStore and MySQL) were installed on VMware virtual machines on several computers with different hardware performances, each with 4 GB and 8 GB RAM allocated to the virtual machines. A test database (composed of four "fact" and several "dimension" tables) was installed on each DBMS and the same queries were executed on the "fact" tables with 5M and 300K rows of each database.

As was to be expected, the fastest execution speed was achieved using the most powerful computer in the experiment executing the commercial DBMSs, HP Vertica and MS SQL Server, both with the database tables indexed in ColumnStore format. The execution of queries Q1 and *Q*3 on larger "fact" tables (with 5M rows) was faster in HP Vertica when the amount of RAM allocated to each VM was 8 GB, but the execution of query *Q*1 was faster on MS SQL Server when 4 GB was allocated to the VM. Execution of the queries *Q*2 and *Q*4 with smaller "fact" tables was faster in HP Vertica regardless of the amount of RAM allocated to the VMs.

The execution speed was also very good on the non-commercial DBMSs, MariaDB CS (ColumnStore) and MySQL DB with the InfoBright storage engine, both using ColumnStore indexing. Queries *Q*1 and *Q*2 executed faster in Oracle 12c than in MariaDB CS, but queries Q3 and Q4 executed faster in MariaDB CS. All queries executed on all computers using Virtual Machines in under 10 seconds, except for the databases using RowStore indexes in MS SQL and MySQL. When installed on hardware directly (not through the VM) on PowerBlade or equivalent servers, the query execution would be even faster.

Based on the results achieved, ColumnStore databases are highly recommended for online querying for analytical purposes.

Acknowledgements

This work has been supported by the Ministry of Science and Technological Development of Republic of Serbia under Project No. TR-35026

7 REFERENCES

[1] Istrat V., Stanisavljev S., & Markoski B. (2015). The role of business intelligence in decision process modelling. *The European Journal of Applied Economics*, 12(2), 44-52. https://doi.org/10.5937/ejae12-8230
[2] DB-Engines Ranking of Relational DBMS. See https://db-engines.com/en/ranking/relational+dbms
[3] TPC Benchmarks & Benchmark Results. See http://www.tpc.org
[4] Osman R. & Knottenbelt W. J. (2012). Database system performance evaluation models: A survey. *Performance Evaluation*, *69*, 471-493. https://doi.org/10.1016/j.peva.2012.05.006
[5] Seng J-L., Yao S. B., & Hevner A. R. (2005). Requirements-driven database systems benchmark method. *Decision Support Systems*, *38*, 629-648. https://doi.org/10.1016/j.dss.2003.06.002
[6] Shanker U., Misra M., & Sarje A. K. (2018). Distributed real time database systems: background and literature review. *Distributed and Parallel Databases*, *23*, 127-149. https://doi.org/10.1007/s10619-008-7024-5
[7] Gu G., Li Q., Wen X., Gao Y., & Zhang X. (2012). An Overview of Newly Open-Source Cloud Storage Platforms. *IEEE International Conference on Granular Computing*. https://doi.org/10.1109/GrC.2012.6468625

[8] Kohler J. & Specht T. (2014). Vertical Query-Join Benchmark in a Cloud Database Environment. *2014 Second World Conference on Complex Systems (WCCS)*., Morocco, https://doi.org/10.1109/ICoCS.2014.7060950
[9] Tongkaw, S. & Tongkaw, A. (2016). A Comparison of Database Performance of MariaDB and MySQL with OLTP Workload. *2016 IEEE Conference on Open Systems (ICOS)*. https://doi.org/10.1109/ICOS.2016.7881999
[10] Yao, S. B., Hevner, A. R., & Young-Myers, H. (1987). Analysis of Database System Architectures Using Benchmarks. *IEEE Transactions on Software Engineering*, SE-13(6), 709-725. https://doi.org/10.1109/TSE.1987.233476
[11] Tudorica, B. G. & Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. *2011 RoEduNet International Conference 10th Edition: Networking in Education and Research*. https://doi.org/10.1109/RoEduNet.2011.5993686
[12] Zafar, R., Yafi, E., Zuhairi, M. F., & Dao, H. (2016). Big Data: The NoSQL and RDBMS review. *2016 International Conference on Information and Communication Technology (ICICTM)*. https://doi.org/10.1109/ICICTM.2016.7890788
[13] Youssef, B. (2012). A Comparative Study on the Performance ofthe Top DBMS Systems. *Journal of Computer Science & Research*, *1*(1), 20-31.
[14] Amlanjyoti, S. et al. (2015). Comparative Performance Analysis of MySQL and SQL Server RDBMSs in Windows Environment. *International Journal of Advanced Research in Computer and Communication Engineering*, *4*(3).
[15] Khawar, I. (2017). Huge and Real-Time Database Systems: A Comparative Study and Review for SQL Server 2016, Oracle 12c & MySQL 5.7 for Personal Computer. *Journal of Basic & Applied Sciences*, *13*, 481-490.
[16] Khalid, M. (2016). Performance Comparison of NOSQL Database Cassandra and SQL Server for Large Databases, *Journal of Independent Studies and Research - Computing*, 14(2).
[17] Kyurkchiev, H. & Kaloyanova, K. (2013). Performance Study of Analytical Queriesof Oracle and Vertica. *Proceedings of Conference Information Systems & Grid Technologies*.
[18] Rotsnarani, S. (2017). Performance Comparison between Apache Hive and Oracle SQL for Big Data Analytics, *Proceedings of International Conference on Soft Computing and Pattern Recognition, Advances in Intelligent Systems and Computing*, 614.
[19] Lamb, A. et al. (2012). The Vertica Analytic Database: CStore 7 Years Later. *Proceedings of the VLDB Endowment*, 5(12).
[20] Chakkappen, S. Optimizer Transformations: Star Transformation. See https://blogs.oracle.com/optimizer/optimizer-transformations:-star-transformation
[21] Oracle 11g R2 Database Data Warehousing Guide. See https://docs.oracle.com/cd/E11882_01/server.112/e25554/toc.htm
[22] Oracle Database 12C In-Memory Option. See http://www.oracle.com/us/solutions/sap/nl23-db12c-imo-en-2209396.pdf
[23] In-Memory CS in Oracle DB 12c Release 1 (12.1.0.2). See https://oracle-base.com/articles/12c/in-memory-column-store-12cr1
[24] Per-Åke, L. et al. (2011). SQL server column store indexes. *SIGMOD'10*.
[25] Column store Indexes Described. See https://msdn.microsoft.com/enus/library/gg492088%28v=sql.120%29.aspx
[26] MariaDB Column Store. See https://mariadb.com/kb/en/library/mariadb-columnstore/
[27] Enterprise Data Warehousing with MySQL. See https://www.scribd.com/doc/3003152/Enterprise-Data-Warehousing-with-MySQL
[28] Schumacher, R. Data Warehousing with MySQL and Infobright. See http://download.nust.na/pub6/mysql/tech-resources/articles/datawarehousing_mysql_infobright.html
[29] Ilic, S. et al. (2018). Comparison of Performances of Built-In Tools for Load and Export of Data to and from Popular Databases. *Computer Science and Technologies*, 1/2018, 32-38.

Contact information:

Slaviša ILIĆ, PhD student
University Singidunum,
Danijelova 32, 11000 Belgrade, Serbia
E-mail: islave@live.com

Siniša ILIĆ, PhD
Faculty of Technical Sciences,
Knjaza Miloša 7, 38220 Kosovska Mitrovica, Serbia
E-mail: sinisa.ilic@pr.ac.rs

Ivan Milovanović, PhD
University Singidunum,
Danijelova 32, 11000 Belgrade, Serbia
E-mail: imilovanovic@singidunum.ac.rs

Petar SPALEVIĆ, PhD
(Corresponding author)
Faculty of Technical Sciences,
Knjaza Miloša 7, 38220 Kosovska Mitrovica, Serbia
E-mail: petar.spalevic@pr.ac.rs

Dragiša MILJKOVIĆ, PhD student
Faculty of Technical Sciences,
Knjaza Miloša 7, 38220 Kosovska Mitrovica, Serbia
E-mail: dragisa.miljkovic@pr.ac.rs