On the Performance of Three In-Memory Data Systems for On Line Analytical Processing

Ionuț HRUBARU, Marin FOTACHE
Al. I. Cuza University of Iași
ionut.hrubaru@gmail.com, fotache@uaic.ro

In-memory database systems are among the most recent and most promising Big Data technologies, being developed and released either as brand new distributed systems or as extensions of old monolith (centralized) database systems. As name suggests, in-memory systems cache all the data into special memory structures. Many are part of the NewSQL strand and target to bridge the gap between OLTP and OLAP into so-called Hybrid Transactional Analytical Systems (HTAP). This paper aims to test the performance of using such type of systems for TPCH analytical workloads. Performance is analyzed in terms of data loading, memory footprint and execution time of the TPCH query set for three in-memory data systems: Oracle, SQL Server and MemSQL. Tests are subsequently deployed on classical on-disk architectures and results compared to in-memory solutions. As in-memory is an enterprise edition feature, associated costs are also considered.
Keywords: *In Memory Databases, OLAP, Analytical Workload, TPCH*

1 Introduction

In terms of data persistence and processing, Big Data systems [1] [2] [3] encompass a broad variety of technologies such as NoSQL data stores [4] [5] [6], Hadoop ecosystem [7] [8] and New SQL [9].

In-memory distributed systems [10] are one of the most recent development of Big Data technologies. They are meant to close the gap between OLTP and OLAP workloads into a single system by offering real time distributed processing and analytics. Some of them are part of the New SQL strand. They rely on a distributed cache system to make data processing faster by limiting the I/O disk bottleneck. As memory price has constantly decreased over time, an array of database technologies has emerged on the market to take advantage of the *in-memory* structures. *In-memory* technologies manifest either as new brand distributed systems (e.g. Spark [11], MemSQL [12], Apache Ignite [13], Geode [14], VoltDB [15]) or as a set of features added to classical relational database systems (Oracle, Microsoft SQL Server and MySQL implement in-memory features in their enterprise editions).

The main promise of *in-memory* persistence concerns the data retrieval and processing speed. Currently the trade-offs come from the lack or poor implementation of essential functionalities such as high availability, storage and transactions. Additionally, cost could raise serious concerns when adopting an *in-memory* solution.

In-memory data systems have a broad range of use cases, from OLTP (On Line Transactional Processing) to OLAP (On Line Analytical Processing) and even a mixture between the two – HTAP (Hybrid Transactional Analytical Processing).

This paper investigates data load performance, memory footprint and query performance of three in-memory systems – MemSQL, Oracle and Microsoft SQL Server. TPC-H benchmark [16] database was used for testing the data loading and query performance. Data was randomly generated using dbGen tool [17] on various scale factors (database loadings). Query performance was assessed (on each database scale factor) by collecting execution duration for the 22 queries in the official set provided by TPC-H.

2 In-Memory Database Systems - Current Products and Research

Moving computation from CPU to memory has gained considerable interest in recent years as a solution for overcoming the bandwidth and latency bottleneck through

DOI: 10.12948/issn14531305/21.1.2017.01

releasing the CPU from some of its tasks [18]. In this new paradigm, the memory chips have both storage and computation capability. Great benefits can be achieved through parallelism in the form of *in-memory* clusters which use GPU (Graphical Processing Unit) power (e.g. Kinetica [19] or SQream [20]).

Mian [21] explores the results of two *in-memory* data systems deployed for healthcare big data management, i.e. MemSQL and VoltDB. The case study focuses on the support for detecting medical fraud, diagnosing diseases at an early stage and generating actionable insights for patients, providers and physicians. Healthcare datasets are large in volume and unstructured, making ad-hoc querying painfully slow, so that *in-memory* systems came as a natural solution. The paper argued that VoltDB performs slower than MemSQL when returning high amounts of distinct rows. Results were inconclusive when queries were simpler and returned a small amount of rows. Also, no details were provided concerning the data loads, testing methodology and results gathering.

Sen et al. [22] describe MemSQL optimization techniques for complex analytical queries (requiring real-time answer) based on heuristics that generate execution plans. The cost-based optimizer can use either a left-deep tree where the result of a join is used as an outer input for the next join or a right-deep tree where the result of a join is used as an inner input to the next join. The latter are called b*ushy plans* and are generated via query-rewrite. The effectiveness of these techniques is analyzed against TPC-H and TPC-DS [23] queries.

Chen and Chiang [24] consider in-memory features an essential capability for BI platforms and Big Data analytics. The paper points out the RDBMSs limitation when handling semi-structured data. A distributed processing environment (e.g. Hadoop) seems more appropriate for ad-hoc extraction, parsing, indexing and analytics.

Goel [25] gives insights on SAP Hana architecture as a system targeting large scale analytics over real time data (high performance OLAP and OLTP workloads simultaneously). It is also ACID compliant and ensures snapshot isolation through MVCC (multi version concurrency control, based on timestamp). The paper insists on the main challenges for enterprise data management: (1) data systems must provide OLTP support so that real-time changes to data are automatically propagated in the queries and (2) data systems must scale to handle huge amount of data and to support OLAP workloads. The decoupling of core database components (e.g. query processing, concurrency control, and persistence) is advocated. This design is made possible by advances in high throughput low-latency networks and storage devices.

Plattner [26] analyses the possibility of using *in-memory* column store databases for OLTP systems. The author argues that OLAP system would beneficiate the most out of columnar storage (by accessing less data compared to row storage). Also, columnar storage could prove suitable for update-intensive applications.

3 Three In-Memory Systems under Scrutiny

NewSQL databases are similar to relational databases in guaranteeing the ACID property of a transaction and offering higher performance for reads and writes. They are also similar to the NoSQL systems as they often claim to offer enhanced performance and integration for both OLAP and OTLP. NewSQL systems are often *in-memory* databases (IMDB). IMDB data systems are mainly designed to:

- Accelerate information storing, retrieving and sorting by holding all records in the main memory.
- Use data structures especially designed for RAM, so that there is no need to maintain and synchronize cached copies of data.
- Provide enough speed for bridging the OLAP- OLTP gap and remove the need for data pre-aggregation (summaries, cubes, etc.)
- Enable real-time analytics and situation awareness on live transactional data.

DOI: 10.12948/issn14531305/21.1.2017.01

- Support ACID (atomic, consistent, isolated and durable) transactions, multi-user access, event triggering, notifications while implementing the industry-standard SQL.
- Simplify internal optimization algorithms and execute fewer CPU instructions (relative to the disk-optimized data systems).
- Generate better execution plans through memory data structures (as opposed to data blocks caching specific to RDBMS)

In memory data stores use either a row storage or a columnar storage to cache the data. For instance, MemSQL uses the row format, Oracle uses columnar storage while SAP Hana keeps in memory a dual format. In a row store, new data can be easily added. However, as a drawback, loading data leads to reading irrelevant blocks (as the data block stores all the record columns). In a columnar store, only the needed data is read. Columnar storage requires less RAM since it could be compressed. It is well suited for analytical queries (GROUP BY). At the same time, decompressing the data will increase the CPU workload as reading more columns would involve more seeks.

The three system under scrutiny in this case study are SQL Server, Oracle and MemSQL. While the first two are RDBMSs with in memory features, MemSQL is a full-fledged IMDB.

As its official documentation states [27], MemSQL is a distributed relational database for both concurrent transactions and large analytics. It uses SQL as the query language with no major differences from the popular RDMBS implementations. It has a simple and understandable architecture composed of two types of nodes:

- Aggregator nodes: store the metadata of the distributed system, route queries and aggregate results;
- Leaf nodes: store data and execute SQL queries issued by the aggregator. A leaf is a MemSQL server instance consisting of multiple partitions. Each partition is a database on that server. Communication between leaves and aggregators is implemented via SQL.

Oracle Database 12c Release 1 Enterprise Edition provides an *In-Memory* Column Store that is part of the SGA, adjustable though INMEMORY_SIZE initialization parameter. Columns, tables, partitions and materialized views are stored in memory using a columnar format that can be enabled at the tablespace level.

Microsoft SQL Server 2014 Enterprise Edition released an in-memory feature optimized for OLTP. It is integrated within the database engine. Since this is optimized for OLTP, it is suited for specific types of workloads. There are two concepts related to this feature: (1) memory optimized tables (tables which are fully stored in memory) and (2) natively compiled stored procedures used to access them. Memory-optimized tables store their data into memory using multiple versions of each row. This technique is characterized as 'non-blocking multi-version optimistic concurrency control' and eliminates both locks and latches. It can thereby achieve significant performance advantages.

Both Oracle and Microsoft solutions are available only with the enterprise edition of the database servers. Consequently, companies must compare licencing costs with performance improvements for determining the feasibility of migrating to an *in-memory* architecture.

4 Methodology, Platforms, Tools

Analysis of the Oracle, SQL Server and MemSQL *in-memory* features was performed on the TPCH schema, data and queries, which is a typical analytical workload [16]. Several criteria were considered:

- Three scale factors: 1GB, 5 GB and 10 GB (60 million rows in LINEITEM table);
- Data load speed;
- Table size in memory - this will determine the capacity planning and the licensing costs;
- Query execution time - TPCH provides 22 queries, each addressing a particular business scenario;

DOI: 10.12948/issn14531305/21.1.2017.01

- Queries are executed on a single node machine - since Oracle and SQL Server cannot distribute a query by default on a cluster;
- Costs;
- The queries were also executed (and results compared) on classical disk storage.

The purpose is to compare the performance of the three technologies; this will indicate whether the cost of upgrading to enterprise edition is justified.

The main characteristics of the system under testing (SUT) were:

- Windows Server 2012 R2 Datacenter (64 bit) operating system
- 32 GB RAM;
- Intel Xeon CPU X5675 @ 3.07 GHz, 6 cores, 12 logical processors;
- Broadcom BCM5709S NetXtreme II GigE (network);
- IBM-ESXS MBF2300RC SCSI Disk.

TPCH schema was created adapting the SQL syntax for each system. Since single node setup was used, tables did not have to be created in a replicated fashion. Tables were created using in memory options specific to each platform.

As already mentioned, TPCH consists of 22 analytical, reporting queries, designed to answer business questions. The data model consists of 8 tables and addresses a sales use case. The queries have an SQL template, with placeholders for the random input parameters coming as inputs from the generated data. The TPCH documentation describes the queries in detail [16].

In SQL Server, the tables were created using MEMORY_OPTIMIZED clause. The durability parameter was set to SCHEMA_AND_DATA to guarantee full transactional consistency (that is expected in a relational database). It is required that each memory optimized table must have at least one associated index. Hence, a non-clustered index was created on the primary key (since most of the queries use inequality operators). On the other hand, hash indexes are recommended for queries containing mainly equality operators (which is not the case for TPCH queries). The amount of memory was increased for the SQL Server instance (a vendor recommendation).

TPCH data was generated using DBGen tool [17] for all three scale factors. The data was loaded in SQL Server using BULK INSERT. After the data loading, SQL Server automatically gathers statistics about the tables. It also builds histograms with data distribution that can help the cost-based optimizer in generating better execution plans. After data loading, tables were checked for being pinned into memory. Cache was cleared (using DBCC commands) before each query run within the same scale factor. *SET statistics IO* was used to register the execution times, and the queries were executed in SQL Server Management Studio. The number of records returned was relatively small (up to 2000) and both the client tool and the database server were installed on the same local machine. Consequently, the results are not supposed to be influenced by the network traffic. In order to free the memory, SQL Server was rebooted after the run of each scale factor query set.

All TPCH queries were generated with QGen utility [16] and the syntax was adapted to match SQL Server ANSI implementation.

Before 12c enterprise edition, Oracle provided *pin table* option, i.e. a table could be cached in memory. *Pin table* was an incipient solution to bypass the disk and cache the blocks into memory. The execution plans generated by the optimizer were unchanged.

In Oracle 12 EE *in-memory* feature is integrated within the database engine. The columnar storage can be compressed.

The cost-based optimizer fully manages the *in-memory* data and generates new types of optimized execution plans. Since it is column-oriented, Oracle *in-memory* is better suited for OLAP workloads, unlike SQL Server, which seems to target mainly the OLTP systems.

Oracle 12c introduces a new way of handling databases, using a container database (*cdb*) and multiple pluggable databases (*pdb*). A local user was created in a pluggable database and special permissions were granted. Database memory was increased to account for *in-memory* tables. Tables were created

DOI: 10.12948/issn14531305/21.1.2017.01

using IN MEMORY option set on *critical*, so that whenever the database is restarted, the tables will still be cached. SQL Developer was used for interacting with the database. The data and queries have been created with the same tools, dbGen and QGen. Data was loaded with *sqldr* utility. A control file (*ctl*) was built for each table to specify the table attributes. The control file was further used as parameter in the *sqldr* utility. All the commands were grouped in a batch file which was launched to load the data. After data loading, the tables were accessed through a SQL query for warming up the cache. Otherwise some of the data would not be pinned into the memory. This can be checked with some dictionary views, such as *v$im_segments.* Statistics were gathered manually by issuing a DBMS_GATHER_STATS command (in SQL Server this task was performed automatically). TPCH query syntax was adapted to Oracle SQL dialect.

MemSQL uses an *in-memory* row store and a disk columnar storage for analytical workloads that do not fit into memory. MemSQL runs on Linux only, so the setup procedure is different from its counterparts. After the installation, a web UI (User Interface) console called *MemSQL Ops* is available for administration. Database creation script and load were customized to support MemSQL syntax. MemSQL is MySQL compliant. For connecting to the MemSQL server (which deploys one leaf node and one coordinator), the MySQL client was installed and then used to create databases. For query execution, MySQL Workbench tool was used.

5 Results

This experimental study was performed targeting four main questions:

- What are the loading times for all scale factors?
- How much memory is required by the data?
- Which platform performs better, under default settings and with no performance tuning applied?
- Does the performance gain of *in*-memory options justify costs of upgrading to enterprise editions?

The discussion will reveal some particularities of each platform and also will point out the bottlenecks observed during testing. Queries were executed 10 times each and an average was calculated to ensure the consistency of the tests.

Data load, memory footprint and query execution

Table load is performed in SQL Server using BULK INSERT command. Load time increased with the scale factor (the number of records in each table).

The .csv files containing 1 GB of data (scale factor = 1) took 1.5 GB of RAM in SQL Server. Increasing by 50% the required space proves that SQL Server does not properly optimize in-memory data structure. Consequently, the memory requirements are higher for this database server (and so are the costs).

When it comes to query execution, it was noticed that the first execution of the query took much longer than the subsequent ones. The explanation lies in the parse and compile phase of the execution which is longer for first run because of building the query execution plan.

Also noticeable was query Q22 did not complete for scale factor of 5GB which came as a surprise considering that for 10GB it was executed successfully in 20 seconds at the first run. On further examination it appears that the optimizer did not succeed in generating an execution plan for the query. This proves the instability of the *in-memory* feature, a consequence of its immaturity. Other queries that took longer time were Q1, Q12 and Q19 which were affected by the parsing time. Subsequent runs completed very fast compared to the first runs. SQL Server scaled well when increasing the scale factor. Execution duration increased accordingly within the same ratio from SF1 to SF5 and SF10.

In Oracle, data was loaded with *sqldr* utility. Loading time was correlated with the scale factor. Memory usage was around 650 MB for

DOI: 10.12948/issn14531305/21.1.2017.01

1 GB of raw (.csv) data which shows that Oracle better optimizes its in-memory structures. This a consequence of the compression Oracle applies for columnar storage.
Also, by contrast to SQL Server, Oracle did not manifest any problem when generating the query execution plan. There were no major differences between duration of each query's first and subsequent runs. Oracle has obviously consistent and predictable execution times.
Query duration increases proportionally with the scale factor, from SF1 to SF5 and then SF10.
However, Oracle manifested problems with the correlated subqueries for which execution time was very inconsistent. One example is Q20 (find suppliers in a particular nation having selected parts that may be candidates for a promotional offer) [16]. When executed first two runs it took 66-67 seconds, while further tun took 11-12 seconds. This was reproduced consistently, no matter how many times the test was repeated, even with server restarted.
Of the three systems, MemSQL is the least optimized in terms of memory footprint. Loading 1 GB of raw (.csv) data required almost 3 GB of RAM. This loading ratio RAM/raw data) of 3 was recorded also for SF5. In the case of SF10, MemSQL could not load the data for Lineitem table, although the machine had a fairly large amount of memory (32GB). If the loading ratio were the same, it would probably have taken at least 30 GB of RAM to load all the data. The error thrown was*: Error Code: 1712. Leaf Error (85.122.22.25:3307): Not enough memory available to complete the current request. The request was not processed.* Also MemSQL started displaying memory error messages when the buffer filled up and.
As for SQL Server, in MemSQL the subsequent query executions were much faster (for which only fetch times are reported). Q11 for SF5 yielded incorrect results and a disproportionate long execution time. The root cause was identified in the HAVING clause which used a subquery and could be reported as a MemSQL bug.
When it comes to scalability, MemSQL performs well. Most of the queries did not show five-times increase in execution time when moving from SF1 to SF5.

6 Discussion

Analysis of the systems under scrutiny was performed on three axes: (1) data load, (2) memory usage and (3) query execution time.
Figure 1 compares data load timings for each *in-memory* platform.

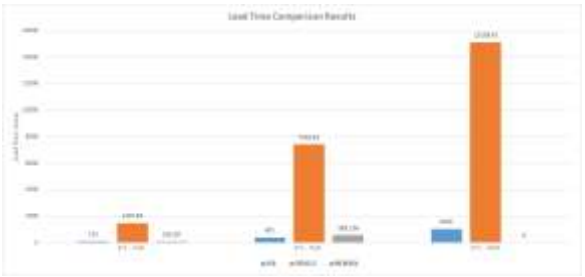

Fig. 1. Load time comparison between platforms

It is evident that SQL Server performed best. MemSQL reported also good results, as long as the data fitted into memory. On a 32 GB machine, SF 10 data could not be loaded because of memory exhaustion. This is a consequence of the MemSQL approach to *in-memory* data storage. Cost would raise spectacularly as MemSQL offers licences per number of GB of RAM, and not CPU cores.
In Oracle data loading took surprisingly long time. That could be explained by the fact that *sqldr* utility does not parallelize the load and does not use all the available CPU cores. External tables could prove a more viable solution for loading the data.
When it comes to scalability, this is achieved by all the three platforms, data load duration being correlated with the scale factor.
In terms of table size in memory (Figure 2), Oracle performs better than SQL Server and MemSQL. This could be explained by Oracle's compressed columnar storage whereas the other two are using a row storage. As previously mentioned, MemSQL requires a lot more RAM for the loaded data.

DOI: 10.12948/issn14531305/21.1.2017.01

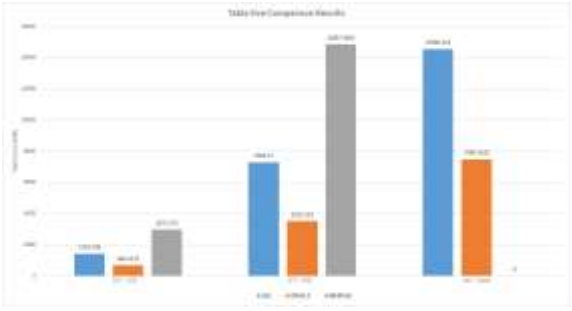

Fig. 2. Memory footprint on each platform

Moving forward to the analysis of query execution time, it was expected that Oracle would performs better for analytical queries (because of the columnar data format). On SF1 Oracle performed best for all the queries, except Q1, Q18 and Q21 where MemSQL outperforms it and also Q4 where SQL Server has the best performance as seen on Figure 3.

Fig. 3. Query execution time for SF 1

Moving to SF 5, MemSQL started to show its strength outperforming SQL Server and matching Oracle in most of the cases, except for a couple of notable cases which have already been discussed: Q11 where it gave incorrect results after a long phase of parsing, Q13 and Q18. Interestingly, SQL Server and Oracle had issues with Q1, while MemSQL managed to complete it a lot faster. Figure 4 shows execution times for SF 5.

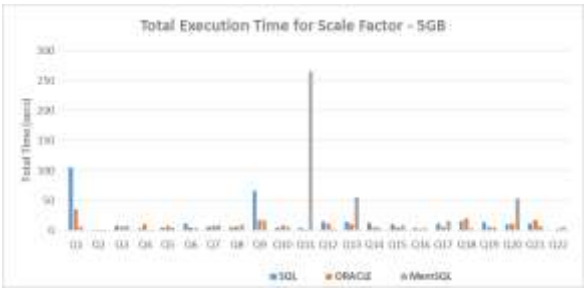

Fig. 4. Query execution time for SF 5

For SF 10, MemSQL does not appear in the results because the data did not fit into memory. The results were mixed between SQL Server and Oracle, as seen in Figure 5.

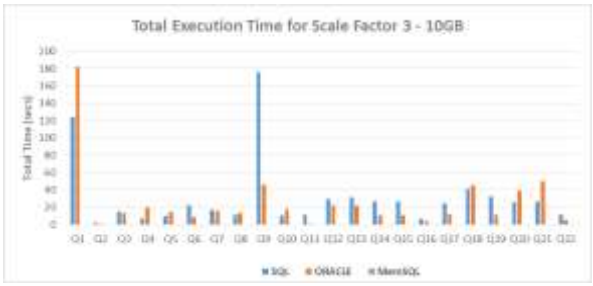

Fig. 5. Query execution time for SF 10

A final chart (Figure 6) combines total execution time for all queries and scale factors. Q11 was removed from the chart since it witnessed a MemSQL bug. Analogously, Q22 was removed for SQL Server on SF 5 because the query could not be completed. The chart confirms that Oracle did perform better than SQL Server on all scale factors. On SF1 Oracle performed better than MemSQL. For SF 5 MemSQL did better, probably because of its latch-free *in-memory* structures, but it remained behind Oracle. One can speculate on what will happen on larger scale factors. SQL Server seems to lag the other two platforms. Results could prove different when using the most recent release, SQL Server 2016, since *in-memory* feature was enhanced and it beneficiates from a combination between *in-memory* row and column storage. The chart also suggests that when scale factor increased, the gap between Oracle and SQL Server reduced, so it would be interesting to test larger database sizes such as 100 GB, or even 1TB.

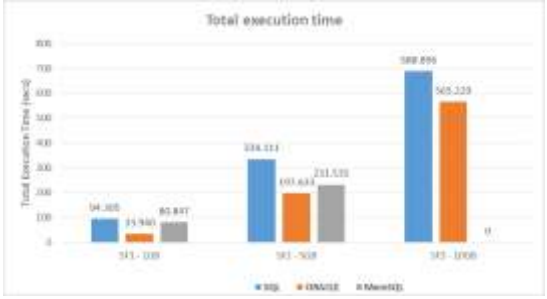

Fig. 6. Total execution time on all platforms and every scale factor

DOI: 10.12948/issn14531305/21.1.2017.01

Decision about upgrading to the enterprise editions should consider the cost. Table 1 displays a cost comparison between standard and enterprise edition (prices are in dollars). They do not include software update license and support which involve additional costs.

Table 1. Enterprise vs standard edition price

	SQL Server	**Oracle**	**MemSQL**
Standard Edition	$3717 per core (in 2 core packs)	$17500 per core	Free (cannot be used in production)
Enterprise Edition	$14256 per core (in 2 core packs)	$47500	$1000 per GB of RAM

MemSQL does not provide a standard edition per se. The freely available Community edition is not recommended to be used in production environments.

Further, the same tests were deployed for comparing *in-memory* versus on-disk storage for each platform. No indexes or any other optimization techniques were considered for on-disk storage tests.

With two exceptions, all queries took less time to be completed on *in-memory* systems. Surprisingly, *on-disk* configuration performed better for Q1 and Q9 in SQL Server. *In-memory* proved to be two to fifteen times faster than *on-disk*, with an average of 2.63 as seen in Figure 7.

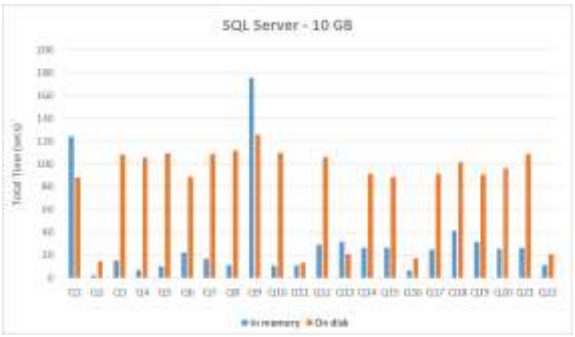

Fig. 7. SQL Server in-memory vs. on disk performance

In Oracle only Q1 was completed faster on disk, as Figure 8 shows. That could be explained by the larger number of various aggregate functions the query contains.

Fig. 8. Oracle in-memory vs on disk performance

Figure 9 presents an overall comparison of the query duration between *in-memory* and *on-disk*.

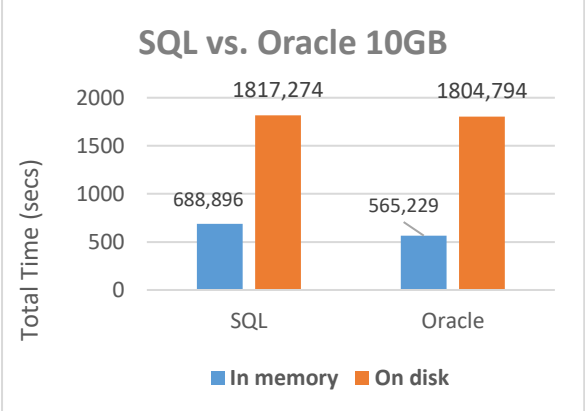

Fig. 9. Total time in-memory vs on disk

Results show a close performance between Oracle and SQL Server especially in the case of *on-disk* storage. Relative to *on-disk*, *in-memory* performance improved by a factor of 3.

Given the cost of Enterprise Edition for both technologies which is three to four times higher, the performance gain might not be feasible.

Currently *in-memory* feature seems to be a rather immature technology and more of a marketing buzz. Nevertheless, it looks promising on the long run.

7 Conclusions and Further Research

For the OLAP workload, the 22 TPCH queries were tested against different scale factors (SF1, SF5 and SF10). Oracle, SQL Server and

DOI: 10.12948/issn14531305/21.1.2017.01

MemSQL were the tested technologies for three separate scenarios: loading data, memory usage and execution times. For loading data into memory, Oracle had the worst performance, which requires a different approach such as external tables. Loading the data should not impact the daily business activities since it can be performed out of the office hours. Consequently, the data processing could beneficiate from the *warm* cache for optimal performance.

The size of required memory for the data revealed that MemSQL consumes three times more RAM than the other two platforms, an effect of *in-memory* data structures design.

The results of TPCH queries run revealed best results for Oracle. MemSQL could not be tested for all scale factors because SF10 data could not fit into memory. For the other two SFs it showed promising results. When comparing with *on-disk* performance, *in-memory* showed a 3x performance increase for both Oracle and SQL Server. Since these are Enterprise Edition options that cost 3-4 times more than the standard editions, companies should carefully evaluate benefits and costs before taking the decision to upgrade.

For future research, *in-memory distributed* systems will be under scrutiny. Many products, both commercial and open source have emerged recently: Apache Ignite, Apache Spark, Geode, Greenplum, Exasol, Sap Hana. They address not only analytical workloads but bridging the gap between OLTP and OLAP into the new hype concept: HTAP.

Also, various optimizations techniques, testing tools and platforms could be used, such as:

- Use schema-only option for SQL Server which will persist data only in memory (not providing durability);
- Compare current results with Oracle *pin-table* option, available in the standard edition;
- Use more nodes and evaluate the MemSQL scalability;
- Create indexes based on predicates from WHERE clauses, JOIN, ORDER BY and GROUP BY;
- Inspect the execution plans and verify the steps that have the highest costs and optimize them;
- Load more data for scale factors up to 100 GB and more;
- Deploy the tests in Amazon cloud to confirm how IOPS (….) and network bandwidth might have an impact;
- Use columnar storage in MemSQL and SQL Server which might prove more appropriate for analytical workloads;
- Use SQL Server 2016 which combines *in-memory* with columnar storage;
- Test performance when increasing the number of concurrent users (e.g. using JMeter);
- Test also OLTP workloads, such as TPC-C.

References

[1] H.U. Buhl, M. Röglinger, and F. Moser, "Big Data: A Fashionable Topic with(out) Sustainable Relevance for Research and Practice?," *Business & Information Systems Engineering*, Vol. 5, Issue 2, 2013, ISSN: 2363-7005, pp.65-69

[2] M. Kowalczyk and P. Buxmann, "Big Data and Information Processing in Organizational Decision Processes," *Business & Information Systems Engineering*, Vol. 6, Issue 5, 2014, ISSN: 2363-7005, pp.267-278

[3] O. Ylijoki, J. Porras, "Perspectives to Definition of Big Data: A Mapping Study and Discussion," *Journal of Innovation Management*, Vol. 4, Issue 1, 2016, ISSN 2183-0606, pp. 69-91

[4] R. Cattell, "Scalable SQL and NoSQL Data Stores," *ACM SIGMOD Record*, Vol. 39, Issue 4, 2010, ISSN 0163-5808, pp. 12-27

[5] D.I. Cogean, M. Fotache, and V. Greavu-Serban, "NoSQL in Higher Education. A Case Study," *Proc. of the 12th International Conference on Informatics in Economy (IE 2013)*, pp.352-360, Bucuresti, Romania

[6] I. Lungu, and B.G. Tudorica, "The Development of a Benchmark Tool for

DOI: 10.12948/issn14531305/21.1.2017.01

NoSQL Databases," *Database Systems Journal*, Vol. 4, Issue 2, 2013, ISSN: 2069-3230, pp.13-20

[7] F. Li, B. C. Ooi, M. T. Özsu, and S. Wu, "Distributed data management using MapReduce," *ACM Computing Surveys*, Vol. 46, Issue 3, Article 31, 2014, ISSN 0360-0300

[8] I. Hrubaru and M. Fotache, "On a Hadoop Cliché: Physical and Logical Models Separation," *Proc. of the 14th International Conference on Informatics in Economy (IE 2015)*, pp. 357-363, Bucharest, Romania, 2015

[9] M. Stonebraker, "New opportunities for New SQL," Communications of the ACM, Vol. 55, Issue 11, 2012, ISSN 0001-0782, pp.10-11

[10] A. Pavlo and M. Aslett, "What's Really New with NewSQL?," *ACM SIGMOD Record*, Vol. 45, Issue 2, 2016, ISSN 0163-5808, pp.45-55.

[11] Apache Spark, http://spark.apache.org/ [December 12, 2016]

[12] MemSQL, http://www.memsql.com/ [December 12, 2016]

[13] Apache Ignite, https://ignite.apache.org/ [December 12, 2016]

[14] Apache Geode https://geode.apache.org/ [December 12, 2016]

[15] VoltDB, https://www.voltdb.com/ [December 12, 2016]

[16] TPC BENCHMARK H (Decision Support) Standard Specification Revision 2.17.1, 2014, Internet: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf [Apr. 10, 2016].

[17] T. Kejser, "Tpch-dbgen Overview," 2014,Internet: https://bitbucket.org/tkejser/tpch-dbgen , [December. 12, 2016].

[18] P. Trancoso, "Moving to memoryland: in-memory computation for existing applications," *Proceedings of the 12th ACM International Conference on Computing Frontiers (CF '15)- Ischia, Italy*, ACM, New York, NY, USA, 2015, Article 32 , 6 pages.

[19] M. Nemschoff, "How to Simplify Streaming Analytics Using a GPU-Accelerated In-Memory Database", Kinetica Blog, http://www.kinetica.com/blog/simplify-streaming-analytics/ [December 12, 2016]

[20] SQream DB GPU-Based SQL Database Technical Overview White Paper, http://www.bigdataleadersforum.com/files/sqream_db_tech_whitepaper_web.pdf [December 12, 2016]

[21] M. Mian, "Healthcare Big Data Exploration in Real-Time", University of Washington, 2014

[22] R. Sen, J. Chen, N. Jimsheleishvilli "Query Optimization Time: The New Bottleneck in Real-time Analytics", ACM, August 2015.

[23] TPC Benchmark DS (TPC-DS), http://www.tpc.org/tpcds/ [December 12, 2016]

[24] H. Chen, R. Chiang, "Business Intelligence and Analytics: From Big Data to Big Impact", MIS Quarterly., Vol 36 No 4, pp. 1165-1188, December 2012

[25] A. Goel et al, "Towards Scalable Realtime Analytics: An Architecture for Scaleout of OLxP Workloads", Proceedings of the VLDB Endowment, Vol. 8, No. 12, August 2015

[26] H. Plattner et al, "A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database", SIGMOD'09, June 29–July 2, 2009 November (2014), pp. 11-22

[27] Official MemSQL documentation, http://docs.memsql.com/docs [December 12, 2016]

DOI: 10.12948/issn14531305/21.1.2017.01

Ionuț HRUBARU graduated *Business Informatics* bachelor programme at Al. I. Cuza of Iași in 2005. He got a master's degree in Business Information Systems. In 2016 he defended his PhD thesis *Polyglot Persistence for Business Applications*. In over 13 years of IT experience Ionut has filled various positions such as programmer, database administrator, and team leader. Currently he is software engineering manager for a local (Iași) IT company. He is particularly interested in Database technology, NoSql, Big Data, leadership and personal development.

Marin FOTACHE has graduated (long time ago) the Faculty of Economics at Al. I. Cuza University of Iasi, Romania. He holds a PhD diploma in Business Information Systems (Business Informatics) from 2000 and he had gone through all didactic positions since 1990 when he joined the staff of Al. I. Cuza University, from teaching assistant in 1990, to full professor in 2002. Currently he is professor within the Department of Accounting, Business Informatics and Statistics in the Faculty of Economics and Business Administration at Alexandru Ioan Cuza University. He is the (co)author of books and journal articles in the fields of knowledge management, SQL, database design, NoSQL, Big Data, and R.

DOI: 10.12948/issn14531305/21.1.2017.01