A comparison of in-memory databases

In the last two months of the summer, the data warehouse team at Tinkoff Bank has been actively engaged in a new discussion topic.

During this time, the team conducted a large-scale test of several in-memory database management systems (DBMS). Virtually every conversation could be started with a DWH administrator asking: «Well, who is in now in the lead?» And you could be sure that it was the right question to ask. The answer, though, was never simple, ending up as a long, emotional tirade about test complications, about communication methods with vendors, some of them unknown before, and about the shortcomings of particular testers.

The goal of the test was to find a fast, analytic in-memory database that fit the bank’s requirements, and to estimate the difficulty of integrating it with the rest of the data warehouse systems. Two DBMS were tested that do not explicitly position themselves as in-memory databases because their caching mechanism works in a way that, when available memory does not exceed data size, can achieve similar performance to “pure” in-memory solutions.

Use case description

The chosen DBMS needs to work as frontend database storage for a selected dataset (with a size of 2-4TB, and the data volume can increase over time): it accepts queries from the BI system (SAP BusinessObjects) and a portion of ad-hoc queries from users. Ninety percent of the queries are SELECT with 1 to 10 equal joins, and some queries with date ranges.

The bank requires that such queries be done much faster than they are now in comparison to Greenplum, the main DBMS currently used for storage.

It is also important that the number of concurrently executed queries does not affect the speed of execution of individual queries — the speed of each query should stay approximately constant. The target DBMS should have the following functionality:

- Horizontal scalability
- Local join support — usage of the «correct» distribution keys for tables
- Column-based storage
- Ability to work with large caches and large amounts of RAM
Data from the main database of the warehouse is to be loaded to the target system, which is why it is very important that the data from the Greenplum DB is loaded quickly and reliably (and preferably incrementally). Integration with SAP BO is also very important. Fortunately, this system can be integrated with almost everything that has stable ODBC-drivers for Windows.

Small, but important desired features include window functions, data duplication (the ability to store several data copies on different nodes), simple cluster extension and parallel data loading.

**Benchmark hardware configuration**

The bank used two physical servers for each database system:

- 16 physical cores (32 with HT)
- 128 GB RAM
- 3.9 TB disk space (RAID 5 with 8 drives)
- 10 Gbit Ethernet Network
- Operating system and operating system settings: according to vendor recommendations

**Benchmark criteria (IC: and prerequisites)**

- Performance of the test queries
- Can be integrated with SAP BO
- Quick and appropriate data import method
- Stable ODBC-drivers
- For non-open-source projects: it was possible to contact the vendor and get the software distribution in a reasonable amount of time

**Benchmarked database management systems**

**Greenplum**

Good old, well-known to us Greenplum. Strictly speaking, Greenplum is not an in-memory DBMS. However, the team found that in experiments, because of XFS features and where it stores its data, it can behave like an in-memory DBMS under certain circumstances.

For example, if there is enough memory for the data available and if the queried data is already located in the memory (cached), disks will not be used at all. Greenplum will instead take all the data from memory. One should understand that this mode of operation is not natural for Greenplum, and application-specific in-memory databases should (in theory) cope better with this task.

For the benchmark, Greenplum was used in the default configuration, without mirrors (primary-segments only). Default settings are used; tables are compressed with zlib.
Yandex Clickhouse

Yandex Clickhouse is a column-based DBMS for analytics and real-time reporting from the famous search giant. The DBMS was installed according to vendor recommendations. For the local tables engine MergeTree, distributed tables were created above the local tables, and these were used in the queries.

SAP HANA

HANA (High-performance ANalytics Appliance) positions itself as a universal tool for analytical and transactional tasks. It has the ability to store its data in a column-based format. It has database system features needed in production, such as disaster recovery, mirroring and replication. HANA allows for flexible configuration of tables partitions (shards) by hash and values intervals.

It supports multilevel partitioning: different partition types can be used on each level. One partition can store up to two billion records.

One of the most interesting features of the database is its table-based configuration of «unload priority», assigning a number from 1 to 10, that shows how likely it is that the table will be unloaded from the physical memory. It gives some flexibility to in-memory and table access speed control: if the table is rarely used, it will obtain the lowest priority. In this case, the table will seldom be loaded into the RAM, and it is moved out of the memory as soon as there is a lack of resources.

Exasol

Exasol is a product that is nearly unknown - a dark horse - in Russia. There is only one big (editor’s note: well-known in Russia) IT company, Badoo, and several non-IT companies that use this DBMS. The full list of the DBMS users can be found on its website.

The vendor promises extraordinarily fast analytics stability akin to a stone in a forest, and the administration simplicity of a coffee machine.

Exasol uses its own operating system — EXAClusterOS (own GNU/Linux distribution based on CentOS/RHEL). Installation of the database is non-standard and unique because it is not an installation of a piece of software on a preconfigured operating system. Instead, an operating system is installed on a separate machine (a virtual machine in our case) from a downloaded ISO-image and only minimal initial configuration is done (such as disk partitioning, network configuration, PXE setup, etc.).

The beauty of the system is that you do not need to install and configure anything (OS, kernel parameters and other "fun" things); adding a node to the cluster is very simple.

You don't need any more than 30 minutes after the server is switched on (bare-metal, without any OS) to the moment when it is up and running as a part of the cluster. The whole configuration and management is done using a web console. The web console is neither overloaded with superfluous functionality, nor at the same time is it too spartan.

Data is stored in a column-based format in memory and is relatively well-compressed (however, we didn’t manage to find any compression configuration). If the database needs more RAM than is available while processing the query, it will start to perform disk swaps (spill). The query will not crash (greetings to HANA and Memsql): only performance is reduced.

Exasol automatically creates and removes indices. If the query is executed for the first time and the database thinks that the query will be processed faster with an index, a new index will be created while processing the query. And if this index was not used during the last 30 days, it will be automatically deleted. What a smart horse!
Memsql

In-memory DBMS based on mySQL. It is a cluster-based DBMS with analytical functions.

By default, it stores data in rows.

To make a column-based store, you need to add a special index while creating the table:

```
KEY `keyname` (`fieldname`) USING CLUSTERED COLUMNSTORE
```

Row store-data is always stored in the memory, but column store-data can be written to the disk if there is not enough memory.

The distribution key is called SHARD KEY. A btree index is created automatically for each shard field key.

The basic version is absolutely free; it has no data and memory limitations. The paid version has several additional features, such as high availability, online backups and restores, data center replications and user rights management.

Impala

Impala is a product of Cloudera; a SQL engine written in C++, it is a part of the Apache Hadoop ecosystem. It works with data stored in HDFS and HBase. To store its metadata it uses HiveMetastore, part of the Hive DBMS. As opposed to Hive, it does not use MapReduce. It supports caching of often-used data blocks.

It is positioned as an analytical DBMS for fast queries. It can work with the most important BI tools. It supports ANSI SQL completely, and it has window functions. Impala is available as a package and a parcel in the Cloudera repository. Cloudera CDH 5.8.0 distribution was chosen for the test. We have chosen the minimal set of services needed by Impala: Zookeeper, HDFS, Yarn, and Hive. Almost all settings used the default configuration. We configured 160 GB of RAM on both servers for Impala. Cgroups was used to control resource utilization by containers.

All optimizations recommended in the article were done, namely:

- Parquet was chosen as the format to store tables in HDFS
- Data types were optimized (everywhere that it was possible)
- Preliminary statistics were gathered for each table (compute stats)
- Table data was cached in HDFS (alter table ... set cached in …)
- Joins were optimized (as much as possible)

At first the team didn’t consider Impala for the test because when they tested it a few years back, it didn’t look production ready. But, convinced that there were crucial changes made in its memory management subsystem in recent years, they determined it would be worth looking at it again.

Additional information about Impala (not translated)

Some important features of the databases were collected in the following Google Docs tables

The same table in the Habr-format (caution! new design of Habr made wide tables unreadable) (not translated)

### Results

Description and text of the queries, used in the benchmarking. Results (in seconds per query)

<table>
<thead>
<tr>
<th></th>
<th>Greenplum</th>
<th>Exasol</th>
<th>Clickhouse</th>
<th>Memsql</th>
<th>SAP Hana</th>
<th>Impala</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>14</td>
<td>&lt; 1</td>
<td>-</td>
<td>108</td>
<td>6</td>
<td>78</td>
</tr>
<tr>
<td>N2</td>
<td>131</td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>127</td>
<td>Error</td>
</tr>
<tr>
<td>N3</td>
<td>67</td>
<td>85</td>
<td>-</td>
<td>-</td>
<td>122</td>
<td>733</td>
</tr>
<tr>
<td>T1</td>
<td>14</td>
<td>1.8</td>
<td>64</td>
<td>70</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>T2</td>
<td>17</td>
<td>4.2</td>
<td>86</td>
<td>105</td>
<td>20</td>
<td>127</td>
</tr>
<tr>
<td>D1</td>
<td>1393</td>
<td>284</td>
<td>-</td>
<td>45</td>
<td>1500</td>
<td>-</td>
</tr>
<tr>
<td>D2</td>
<td>&gt; 7200</td>
<td>1200</td>
<td>-</td>
<td>&gt; 7200</td>
<td>Error</td>
<td>-</td>
</tr>
</tbody>
</table>
Yandex Clickhouse

During the benchmark it became clear that this DBMS does not fit the team’s tasks — there is only nominal joins support. For example:

- Only JOINs with subqueries in the right part are supported
- Joins conditions do not pass in the subquery
- Distributed joins are inefficient

It was almost impossible to rewrite complex queries (N1-N3) in the Clickhouse syntax. There was another disappointing limitation — the result of each query must fit into the memory of one (!) node of the cluster.

Although this DBMS did not fit for the team’s BI tasks, the test has shown that it can be used in other projects.

The documentation was detailed and thorough, accessible on its official site (unfortunately, it does not cover all aspects of the database usage at the moment), and the Yandex developers were available and willing to deliver quick answers to questions that came up during the test.

Exasol

The most important thing we saw with the Exasol database was that it works really fast and is surprisingly stable, and it does this out of the box. In almost all the benchmarks, Exasol offered performance advantages compared to all other DBMS. However, unlike many other DBMS, it is a black box — you can’t even access the nodes using SSH, and start iotop, htop and other similar tools there. The team felt really nervous not being able to control its own servers.

However, it should be noted that all necessary data-related hardware load and database work is available in the system view of the database.

Exasol has JDBC, ODBC drivers; perfect ANSI SQL support and support of some special Oracle features (select 1 from dual, for example). The DBMS has embedded JDBC drivers to access external databases (Oracle, PostgreSQL, MySQL and others), which is very convenient for data loading.

Exasol allows seeing the execution plan only for the queries that were executed already. This is the case because the plan is created just-in-time during the queries’ execution using the analysis of the intermediate results. The team is used to having such a classical explanation, although the lack of it does not hinder work.

The database is fast, convenient, reliable, and doesn’t need much tuning — you can install it and forget about it. So, it’s just exactly what you want. But it is a black box, and that is suspicious.

The team would like to separately emphasize the quality of support: the support service operatively answered all questions that emerged during the team’s experiments.

SAP HANA

The largest part of the installation, configuration and tuning was done by consulting companies who are SAP Partners in Russia. Without their help the team could not have taken a look at the database and rate its advantages. This has proven that you need experience to work with HANA.

It was very interesting to see how HANA counts strings in a self-join table:

```
EXPLAIN of a self-join query in HANA
```

The optimizer evaluates the result using statistics without creating joins at all. Not a bad trick, but if we add a condition to the \`WHERE\` clause, for example \`1 = 1\`, then the trick does not work anymore and joins then took 25 minutes; meaning almost as much time as in Greenplum.

At the moment, HANA can’t store intermediate query results on disks. So, if there is not enough memory, the session terminates and the user doesn’t get any results.

It became clear during the test that even if data of joined tables is distributed over the cluster nodes correctly, join is executed only on one node of the cluster. The join is evaluated inside the node before the execution data from the whole cluster is copied. The team was not able to modify this behavior in the time available for the test to make the join execute locally.

The vendor recommends, where possible, to use one-node database configuration and that is confirmed by the results of the team’s test: it is much more difficult to make a two-node installation work optimally compared to a one-node installation.
Memsql

Installing MemSQL is quick and easy. The administrative web interface is nice but not very smart. A couple of examples: you can add nodes to the cluster, but you can’t remove them (at least not in an obvious way) or you can see the list of running and finished queries, but you can’t find any details about them.

You can tweet the number of the queries per second using the administrative interface of MemSQL; it loves to put processors under high load, and there were almost no errors with memory overflow. Before joining two tables, MemSQL performs repartitioning based on the join key.

In the team’s case, they can store the data_bal and date tables with a complex shard key (account_rk, valid_from_dt); the shard key for the scd-table is (account_rk, scd_valid_from_dt). Join between data_bal and date is very fast in this case. After that, during the query execution, the data will be redistributed by account_rk and scd_valid_from_dt, and on the next step — by account_rk to join with the financial_account table. As the support service states, repartitioning is a very time-consuming operation.

The team’s queries seemed to be too heavy for this DBMS because of too many different joins. In Greenplum, joins between the named tables are done locally, and because of that, are faster; in this case there is no Redistribute Motion.

Generally speaking, MemSQL seems to be a perfect DBMS to migrate from MySQL if no overly complex analytical queries need to be used.

Impala

The installation of the Cloudera cluster contains Impala, which is very simple and well-documented.

It should be noted that the speed of Impala in comparison to the other DBMSes was not very good. The query that calculated count(*) in d_financial_account_not_additive, for example, took 3.5 minutes in Impala. This was much longer than that of the competitors who needed less than dozens of seconds for the same query.

The team also conducted an interesting experiment. The d_financial_account_not_additive view has two joins. There is a join by the account_rk field with integer the datatype and with fields with the date datatype in each of them. Impala does not have the date datatype, so the team had to use the timestamp type instead. After replacing timestamp with bigint with UNIX timestamp stored in it, query execution time improves by a minute.

This improves the query result by one minute. In the next step the team united account_rk and the date fields (valid_from_dt and scd_valid_from_dt), to make join only on one field. It was done using this simple method:

\[ \text{account_valid_from} = \text{account_rk} * 100000 + \text{cast(unix_timestamp(valid_from_dt)/86400 as int)} \]

Joins over one field delivered an improvement of approximately half a minute, but query time was nonetheless several times more than in other DBMSes.

The main queries took several times longer to run. The N2 query failed because of memory limitations, which is why there are no results for it.

At the moment, Impala does not support hash-based data distribution over nodes, which is why probably why no good execution times were achieved for the queries used.

Not a conclusion

The team deliberately does not want to make any conclusions along the lines of «DBMS A can do this and this and DBMS B can tweet from its administrative web interface, and that’s why A is better than B.» The reader should make his own conclusions. We hope that this article will help those who are considering new products in the DBMS for ETL and BI market, and help them to make a final decision regarding the matter.

The article is published on behalf of the authors, who ran the benchmarks:

- Maxim Belousov
- Dmitriy Nemchin
- Georgiy Bezrukiikh
- Dmitriy Pavlov