Collaborative Project

LOD2 – Creating Knowledge out of Interlinked Data

Project Number: 257943 **Start Date of Project:** 01/09/2010 **Duration:** 48 months

Deliverable 2.1.4

150 Billion Triple dataset hosted on the LOD2 Knowledge Store Cluster

Dissemination Level	Public
Due Date of Deliverable	Month 27, 30/11/2012
Actual Submission Date	Month 31, 18/03/2013
Work Package	WP 2, Storing and Querying Very Large Knowledge bases
Task	T2.1.4
Type	Software
Approval Status	Approved
Version	1.0
Number of Pages	14
Filename	LOD2_D2.1.4_LOD_Cloud_Hosted_On_The_LOD2_Knowledge_Store_Cluster_150B_Triples

Abstract:

This report gives an overview of the Virtuoso Column Store Edition Cluster BSBM benchmarking testing against 50 and 150 billion triple generated datasets, to prove the ability to scale these size datasets with multiple concurrent user workloads.

Project co-funded by the European Commission within the Seventh Framework Programme (2007 – 2013)

History

Version	Date	Reason	Revised by
0.1	25/01/2013	Draft of 150B triples LOD Cloud hosted on the LOD2 Knowledge Store Cluster	Hugh Williams Orri Erling Pham Duc
0.2	25/02/2013	Peer Review comments	Renaud Delbru
0.5	26/02/2013	Approval comments	Peter Boncz
0.9	04/03/2013	Final revisions	Hugh Williams Pham Duc Orri Erling
1.0	18/03/2013	Final Review	Renaud Delbru

Author List

Organisation	Name	Contact Information
OGL	Hugh Williams	hwilliams@openlinksw.com
OGL	Orri Erling	oerling@openlinksw.com
CWI	Pham Duc	P.Minh.Duc@cwi.nl
CWI	Peter Boncz	P.Boncz@cwi.nl

Table of Contents

Executive Summary

This report gives an overview of the Virtuoso Column Store Edition Cluster (V7) Berlin SPARQL Benchmark (BSBM) testing against 50 and 150 billion triple generated datasets, to prove the ability to scale to these size datasets with multiple concurrent user workloads.

The Berlin SPARQL Benchmark (BSBM) was chosen to perform this evaluation, as unlike in previous years the LOD Cloud does not contain 150 billion triples to enable its use. The evaluation was performed on the CWI Scilens hardware cluster in Amsterdam.

We note that this experiment marks a 750x increase in reported data size on the BSBM benchmark. It is also the first time that BSBM was run on a cluster system. Further this is the first time that also results on the Business Intelligence workload of BSBM could be reported.

All this marks a strong increase in the state-of-the-art of RDF data management. Important parts of this innovation were the fruit of LOD2 WP2 activities.

Motivation

1.1 Virtuoso Column Store (V7) Cluster Edition

The original Virtuoso Server was developed as a row-wise transactional orientated RDBMS including built-in RDF Data storage , with clustered capabilities for scale out across commodity level hardware enabling the hosting of large RDF datasets in particular. As the RDF Datasets in the LOD cloud generally have grown in size, the need to host these datasets with increasing scale and performance has tested the row-wise RDF implementation to it limits.

Thus, with a view to trying to get RDF data storage and querying on a par with the relational model, as part of the LOD2 project in collaboration with our partner CWI, the compressed column-wise storage and vectorized execution model has been implemented from the BI (business intelligence) world, which is more suited to typical RDF workloads, and enables significant improvements in performance and reduced data storage size.

1.2 BSBM benchmark results to date

The BSBM (Berlin SPARQL BenchMark) was developed in 2008 as one of the first open source and publicly available benchmarks for RDF data stores. BSBM has been improved over this time and is current on release 3.1 which includes both Explore and Business Intelligence use case query mixes, the latter stress-testing the SPARQL1.1 group-by and aggregation functionality, demonstrating the use of SPARQL in complex analytical queries.

Results:

The following BSBM results have been published the last being in 2011, all of which include results for the Virtuoso version available at that time (all but the last one being for Virtuoso row store) and can be used for comparison with the results produced in the deliverable:

- BSBM version 1 (July 2008) – with 100 million triple datasets
- BSBM version 2 (Nov 2009) – with 200 million triple datasets
- BSBM version 3 (Feb 2011) - with 200 million triple datasets

The above results are all for the Explore use case query mix only. Apart from these 'official' BSBM results, in published literature some BSBM results have appeared, though none of these complete BI runs or Explore runs on any larger size.

The results of this deliverable benchmarking the BSBM Explore and BI use case query mixes against 50 and 150 billion triple datasets on a clustered server architecture represent a major step (750 times more data) in the evolution of this benchmark.

Cluster Configuration

RDF systems strongly benefit from having the working set of the data in RAM. As such, the ideal cluster architecture for RDF systems uses cluster nodes with relatively large memories. For this reason, we selected the CWI SCILENS (www.scilens.org) cluster for these experiments. This cluster is designed for high I/O bandwidth, and consists of multiple layers of machines. In order to get large amounts of RAM, we used only the "bricks" layer, which contains its most powerful machines.

Virtuoso V7 Column Store Cluster Edition was set up on 8 Linux machines. Each machine has two CPUs (8 cores and hyperthreading, running at 2GHz) of the Sandy Bridge architecture, coupled with 256GB RAM and three magnetic hard drives (SATA) in RAID 0 (180/MB/s sequential throughput). The machines were connected by Mellanox MCX353A-QCBT ConnectX3 VPI HCA card (QDR IB 40Gb/s and 10GigE) through an InfiniScale IV QDR InfiniBand Switch (Mellanox MIS5025Q). The cluster setups have 2 processes per machine, 1 for each CPU. (A CPU here has its own memory controller which makes it a NUMA node). CPU affinity is set so that each server process has one core dedicated to the cluster traffic reading thread (i.e. dedicated to network communication) and the other cores of the NUMA node are shared by the remaining threads. The reason for this set-up is that communication tasks should be handled with high-priority, because failure to handle messages delays all threads. It was verified experimentally that this configuration works best.

These experiments have been conducted over many months, in parallel to the Virtuoso V7 Column Stre Cluster Edition software getting ready for release. Large part of the effort spent was in resolving problems and tuning the software.

Data Load

The original BSBM data generator was a single-threaded program. Generating 150B triples with it would have taken weeks. As part of this project, we modified the data generator to be able to generate only a subset of the dataset. By executing the BSBM data generator in parallel on different machines, each generating a different part of the dataset, BSBM data generation now has become scalable.

In these experiments we generated 1000 data files with the BSBM data generator. Separate file generation is done using the –nof option in the BSBM driver. These files are then distributed to each machine according to the modulo of 8 (i.e., the number of machine) so that files number 1, 9, 17 … go to machine 1, file number 2, 10, 18,… go to machine 2, and so on. This striping of data of the data across the nodes ensures a uniform load, such that all nodes get an equal amount of similar data.

# of triples	Total file size (in .ttl format)	Total file size (with gzip compression)	Total data size of each machine (in gzip)	Database size (total)	Loading Time
50 billion triples	2.8 TB	240 GB	30 GB	1.8 TB	10h 45s
150 billion triples	8.5 TB	728 GB	91 GB	5.6 TB	n/a

Each machine loaded its local set of files (125 files), using the standard parallel bulk-load mechanism of Virtuoso. This means that multiple files are read at the same time by the multiple cores of each CPU. The best performance was obtained with 7 loading threads per server process. Hence, with two server processes per machine and 8 machines, 112 files were being read at the same time.

Also notice that in a cluster architecture there is a constant need for communication during loading, since all new URIs and literal must be encoded identically across the cluster; hence shared dictionaries must be accessed. Thus, a single loader thread counts for about 250% CPU across the cluster.

The load was non-transactional and with no logging, to maximize performance.

Aggregate load rates of up to 2.5M quads per second were observed for periods of up to 30 minutes.

The total loading time for the dataset of 50 billion triples is about 10h 45 minutes, which makes the average loading speed 1.3M triples per second.

The largest load (150G quads) was slowed down by one machine showing markedly lower disk write throughput than the others. On the slowest machine *iostat* showed a continuous disk activity of about 700 device transactions per second, writing anything from 1 to 3 MB of data per second. On the other machines, disks were mostly idle with occasional flushing of database buffers to disk producing up to 2000 device transactions per second and 100MB/s write throughput. Since data is evenly divided and 2 of 16 processes were not runnable because the OS had too much buffered disk writes, this could stop the whole cluster for up to several minutes at a stretch. Our theory is that these problems were being caused by hardware malfunction.

To complete the 150B load, we interrupted the stalling server processes, moved the data directories to different drives, and resumed the loading again. The need for manual intervention, and the prior period of very slow progress makes it hard to calculate the total time it took for the 150B load. We may re-run the loading for 150 billion triples in the near future and report it when publishing the benchmark results.

Notes on the BI Workload

For running the benchmark, we used the Business Intelligence Benchmark [BIBM], an updated version of the original BSBM benchmark [BSBM] which provides several modifications in the test driver and the data generator. These changes have been adopted in the official V3.1 BSBM benchmark definition. The changes are as follows:

- The test driver reports more and more detailed metrics including "power" and "throughput" scores.
- The test driver has a drill down mode that starts at a broad product category, and then zooms in subsequent queries into smaller categories. Previously, the product category query parameter was picked randomly for each query; if this was a broad category, the query would be very slow; if it is a very specific category it would run very fast. This made it hard to compare individual query runs; and also introduced large variation in the overall result metric. The drill down mode makes it more stable and also tests a query pattern (drill down) that is common in practice.
- One query (i.e., BI Q6) was removed from the Explorer use case that returned a quadratic result. This query would become very expensive in the 1 billion triple and larger tests, so its performance would dominate the result.
- The text data in the generated strings is more realistic. This means you can do (more) sensible keyword queries on it.
- The new generator was adapted to enable parallel data generation. Specifically, one can let it generate a subset of the data files. By starting multiple data generators on multiple machines one can thus hand-parallelize data generation. This is quite handy for the larger-size tests, which

literally otherwise takes weeks.

As the original BSBM benchmark, the test driver can run with single-user run or multi-user run.

- **Single user run**: This simulates the case that one user executes the query mix against the system under test.
- **Multi-user run**: This simulates the case that multiple users concurrently execute query mixes against the system under test.

All BSBM BI runs were with minimal disk IO. No specific warm-up was used and the single user run was run immediately following a cold start of the multiuser run. The working set of BSBM BI is approximately 3 bytes per quad in the database. The space consumption without literals and URI strings is 8 bytes with Virtuoso column store default settings.

For a single user run, typical CPU utilization was around 190 of 256 core threads busy. For a multiuser run, all core threads were typically busy. Hence we see that the 4 user run takes roughly 3 times the real time of the single user run.

BSBM Benchmark Results

The following terms will be used in the tables representing the benchmark results.

- **Elapsed runtime** (seconds): the total runtime of all the queries excluding the time for warm-up runs.

- **Throughput**: the number of executed queries per hour. This value is computed with considering the scale factor as in TPC-H. Specifically, the throughput is calculated using the following function.

  *Throughput = (Total # of executed queries) * (3600 / ElapsedTime) * scaleFactor.*

  Here, the scale factor for the 50 billion triples dataset and 150 billion triples dataset is ~ 500 and 1500, respectively.

- **Timeshare**(%): The percentage of the query runtime in total runtime of all queries.

  *Timeshare(query q) = (Total runtime of q) / (Total runtime of all queries) * 100*

(It also can be calculated as Timeshare(query q) = 100*aqet*runsPerQuery/totalRuntime, where

aqet is the average query execution time for a query, runsPerQuery is the number of runs of that query)

- **AQET: average query execution time** (seconds): The average execution time of each query computed by the total runtime of that query and the number of executions.

  AQET(q) = (Total runtime of q) / (number of executions of q)

1.3 BI use case

Note: No warm-up for both single client and multiple clients while running BI use case

	50B triples				**150B triples**			
	Single-client		**Multi-client (4)**		**Single-client**		**Multi-client (4)**	
runtime	3733sec		9066sec		12649sec		29991sec	
Tput	12052		19851		10671		18003	
	AQET	Timeshare	AQET	Timeshare	AQET	Timeshare	AQET	Timeshare
Q1	622.80	(16.7%)	1085.82	(12.8%)	914.39	(7.2%)	1591.37	(5.6%)
Q2	189.85	(5.1%)	30.18	(0.4%)	196.01	(1.6%)	507.02	(1.8%)
Q3	337.64	(9.0%)	2574.65	(30.3%)	942.97	(7.5%)	8447.73	(30.0%)
Q4	18.13	(2.4%	6.13	(0.4%)	183.00	(7.2%)	125.71	(2.2%)
Q5	187.60	(25.1%)	319.75s	(18.8%)	830.26	(32.8%)	1342.08	(23.8%)
Q6	47.64	(1.3%)	34.67	(0.4%)	24.45	(0.2%)	191.42	(0.7%)
Q7	36.96	(5.9%)	39.37	(2.8%)	58.63	(2.8%)	94.82	(2.0%)
Q8	256.93	(34.4%)	583.20	(34.3%)	1030.73	(40.7%)	1920.03	(34.0%)
Avg	212		584		522		1777s	

Some results seem noisy, for instance Q2@50B, Q4@50B, Q4@150B are significantly cheaper in the multi-client-setup. Given the fact that the benchmark was run in drill-down mode, this is unexpected. It could be countered by performing more runs, but, this would lead to very large run-times as the BI workload has many long-running queries.

In the following, we discuss the above performance results over several specific queries Q2 and Q3.

Query 2 in the BI use case:

```
SELECT ?otherProduct ?sameFeatures
{
  ?otherProduct a bsbm:Product .
  FILTER(?otherProduct != %Product%)
  {
    SELECT ?otherProduct (count(?otherFeature) As ?sameFeatures)
    {
      %Product% bsbm:productFeature ?feature .
      ?otherProduct bsbm:productFeature ?otherFeature .
      FILTER(?feature=?otherFeature)
    }
    Group By ?otherProduct
  }
}
```

```
  Order By desc(?sameFeatures) ?otherProduct
  Limit 10
```

BI Q2 is a lookup for the products with the most features in common with a given product. The parameter choices (i.e., %Product%) produce a large variation in run times. Hence the percentage of the query's timeshare varies according to the repetitions of this query's execution. For the case of 4-client, this query is executed for 4 times which can be the reason for the difference timeshare between single-client and 4-client of this query.

Query 3 in the BI use case:

```
  Select ?product (xsd:float(?monthCount)/?monthBeforeCount As ?ratio)
  {
    { Select ?product (count(?review) As ?monthCount)
      {
        ?review bsbm:reviewFor ?product .
        ?review dc:date ?date .
        Filter(?date >= "%ConsecutiveMonth_1%"^^<http://www.w3.org/2001/XMLSchema#date> && ?date <
"%ConsecutiveMonth_2%"^^<http://www.w3.org/2001/XMLSchema#date>)
      }
      Group By ?product
    }  {
      Select ?product (count(?review) As ?monthBeforeCount)
      {
        ?review bsbm:reviewFor ?product .
        ?review dc:date ?date .
        Filter(?date >= "%ConsecutiveMonth_0%"^^<http://www.w3.org/2001/XMLSchema#date> && ?date <
"%ConsecutiveMonth_1%"^^<http://www.w3.org/2001/XMLSchema#date>) #
      }
      Group By ?product
      Having (count(?review)>0)
    }
  }
  Order By desc(xsd:float(?monthCount) / ?monthBeforeCount) ?product
  Limit 10
```

The query generates a large intermediate result: all the products and their review count on the latter of the two months. This takes about 16GB (in case of 150 billion triples), which causes this to be handled in the buffer pool, i.e. the data does not all have to be in memory. With multiple users connected to the same server process, there is a likelihood of multiple large intermediate results having to be stored at the same time. This causes the results to revert earlier to a representation that can overflow to disk. Supposing 3 concurrent instances of Q3 on the same server process, the buffer pool of approximately 80G has approximately 48G taken by these intermediate results. This causes pages needed by the query to be paged out, leading to disk access later in the query. Thus the effect of many instances of Q3 on the same server at the same time decreases the throughput more than linearly. This is the reason for the difference in timeshare percentage between the single-user and multi-user runs.

The further problem in this query is that the large aggregation on count is on the end result, which re-aggregates the aggregates produced by different worker threads. This re-aggregation is due to the large amount of groups quite costly; therefore it dominates the execution time: the query does not

parallelize well. A better plan would hash-split the aggregates early, such that re-aggregation is not required.

1.4 Explore use case

Note:

+ Single client: 1000 warm-ups and 100 runs

+ 4 clients: 8 warm-ups

	50B triples				**150B triples**			
	Single-client		**Multi-client (4)**		**Single-client**		**Multi-client (4)**	
runtime	931sec (100 runs)		15sec (1run)		1894sec (100 runs)		29sec (1 run)	
Tput	4832691		11820943		7126307		18386227	
	AQET	Timeshare	AQET	Timeshare	AQET	Timeshare	AQET	Timeshare
Q1	0.066	(0.7%)	0.415	(3.1%)	0.113	(0.6%)	0.093	(0.4%)
Q2	0.045	(3.0%)	0.041	(1.8%)	0.066	(2.1%)	0.086	(2.3%)
Q3	0.112	(1.2%)	0.091	(0.7%)	0.111	(0.6%)	0.116	(0.5%)
Q4	0.156	(1.7%)	0.102	(0.8%)	0.308	(1.6%)	0.230	(1.0%)
Q5	3.748	(80.8%)	6.190	(91.4%)	8.052	(85.2%)	9.655	(85.4%)
Q7	0.155	(6.7%)	0.043	(1.3%)	0.258	(5.5%)	0.360	(6.4%)
Q8	0.100	(2.1%)	0.021	(0.3%)	0.188	(2.0%)	0.186	(1.6%)
Q9	0.011	(0.5%)	0.010	(0.3%)	0.011	(0.2%)	0.011	(0.2%
Q10	0.147	(3.2%)	0.020	(0.3%)	0.201	(2.1%)	0.242	(2.1%)
Q11	0.005	(0.1%)	0.004	(0.0%)	0.006	(0.0%)	0.006	(0.0%)
Q12	0.014	(0.1%)	0.019	(0.1%)	0.013	(0.1%)	0.010	(0.0%)
Avg (without Q5)	0.08s		0.08s		0.13s		0.13s	

We notice that these 4-client results seem more noisy than the single-client results and therefore it may be advisable in future benchmarking to also use multiple runs for multi-client tests.

What is striking in the Explore results is that Q5 dominates execution time.

Query 5 in the Explore use case:

```
SELECT DISTINCT ?product ?productLabel
WHERE {
        ?product rdfs:label ?productLabel .
 FILTER (%ProductXYZ% != ?product)
```

```
        %ProductXYZ% bsbm:productFeature ?prodFeature .
        ?product bsbm:productFeature ?prodFeature .
        %ProductXYZ% bsbm:productPropertyNumeric1 ?origProperty1 .
        ?product bsbm:productPropertyNumeric1 ?simProperty1 .
        FILTER (?simProperty1 < (?origProperty1 + 120) && ?simProperty1 > (?origProperty1 - 120))
        %ProductXYZ% bsbm:productPropertyNumeric2 ?origProperty2 .
        ?product bsbm:productPropertyNumeric2 ?simProperty2 .
        FILTER (?simProperty2 < (?origProperty2 + 170) && ?simProperty2 > (?origProperty2 - 170))
}
ORDER BY ?productLabel
LIMIT 5
```

Q5 asks for the 5 most similar products to one given product, based on two numeric product properties (using range selections). It is notable that such range selections might not be computable with the help of indexes; and/or the boundaries of both 120 and 170 below and above may lead to many products being considered 'similar'. Given the type of query, it is not surprising to see that Q5 is significantly more expensive than all other queries in the Explore use case (the other queries are lookups that are index computable – this also means that execution time on them is low regardless scale factor).

In the explore use case, most of the queries have the constant running time regardless of the scalefactor, thus computing the throughput by multiplying the qph (queries per hour) with the scalefactor may show a significant increase between the cases of 50-billion and 150-billion triples.

In this case, instead of the throughput metric, it is better to use another metric, namely qmph (# of query mixes per hour).

50 Billion triples		
	Single Client	Multiple Clients (4 clients)
qmph	4253.157	2837.285

150 Billion triples		
	Single Client	Multiple Clients (4 clients)
qmph	2090.574	1471.032

Conclusion

The present experiments demonstrate that complex queries each touching a large fraction of a data set of 150 billion triples can be run in near interactive time on a cluster of commodity servers. The platform utilization is good, i.e. a single query uses about 2/3 of available resources, in this case a total of 256 cores (hyperthreaded cores). Execution times are multiplied by a factor of three when going from a single client to 4 concurrent clients, which is consistent with a good degree of parallelism within a single query, a necessary prerequisite for complex analytic workloads.

The queries themselves consist almost entirely of cross partition joins, thus we see that scalability does not result from an "embarrassingly parallel" workload, i.e. one where each partition can complete with no or minimal interaction with other partitions.

We also see a linear increase in execution times when tripling the data size from 50 billion to 150 billion triples. The queries are generally in the order of O (n * log (n)), where n is the count of triples. The log component comes from the fact of using tree indices where access times are in principle logarithmic to index size. However the log element is almost not felt in the results due to exploitation of vectoring for amortizing index access cost.

The execution is not significantly bound by interconnect, as we observe aggregate throughputs of about 2GB/s on the 8 node QDR InfiniBand network, whose throughput in a n:n messaging pattern is several times higher. Latency is also not seen to cut on CPU utilization, as the CPU percent is high and execution has minimal synchronous barriers.

Having established this, several areas of potential improvement remain. Some queries produce intermediate results that are all passed via a central location when this is not in fact necessary (Q3 BI. Load on aggregation can be partitioned better by using the GROUP BY key as partitioning key. All the joining in the benchmark, which consists almost only of JOIN's and GROUP BY's was done with index lookups. Use of hash joins in many places could improve both throughput and locality, cutting down on network traffic.

Memory efficiency of the query execution may be improved so as to allow more parallel queries to run at high vector size, which is a central component of performance. Message compression may also reduce blocking on message passing, yielding smoother execution with less forced task switches.

In any case the present results demonstrate that complex query loads on a schema-less data model are feasible at scale.

References

[BSBM] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/

[BIBM] https://sourceforge.net/projects/bibm/

[Virtuoso Colum Store] http://sites.computer.org/debull/A12mar/vicol.pdf