7 Analysis of experiments

Abstract

In this research we have provided series of experiments to identify any trends, relationships and patterns in connection to NL-addressing and its implementations. We have realized three types of experiments:

- *Basic experiments to compare our access method with two main types of organization and access to information – sequential and relational;*
- *Experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of structured datasets;*
- *Experiments to examine the applicability of NL-addressing for middle-size and very large RDF-datasets.*

Below we will analyze these experiments. Special attention will be paid to analyzing of storing times of NL-ArM access method and its possibilities for multi-processing.

7.1 Analysis of basic experiments

We started our experiments in *Chapter 4* with two main types of basic experiments: NL-ArM has been compared with:

- Sequential text file of records;
- Relational data base management system Firebird.

The need to compare NL-ArM access method with text files was determined by practical considerations – in many applications the text files are main approach for storing information. To investigate the size of files and speed of their generation we compared writing in a sequential text file and in a NL-ArM archive.

The experiments have sown:

- NL-ArM is constantly about two and half times slower than writing in sequential text file because of building the hash tables in the NL-ArM archive. This means that including new records in NL-ArM archive take the same time (about 0.013 ms) per record irrespective of the number of already stored records;
- The comparison of file sizes showed that, for great number of elements, text file became longer than NL-ArM archive. This leads to the following conclusion - bulky text files whose records are attached to long “keywords” would be saved more compactly in

NL-ArM archives instead as records in text files with explicitly given keywords in each record.

Important consideration in this case was that reading sequential text file to find concrete keyword is very slow operation. Every indexed approach is quicker. Indexed text files are typical for databases and this case was analyzed in experiments with a database.

To provide experiments with a relational database, we have chosen the system “Firebird”. It should be noted that Firebird and NL-ArM have fundamentally different physical organization of data and the tests cover small field of features of both systems.

Firebird is not access method but system for managing databases. It is important that all queries in Firebird are in SQL and interpreted by the system, which requires appropriate time. The primary goal for new Version 2.5 of Firebird is to establish the basics for a new threading architecture which will speed up the Firebird on multi-processors’ computer systems [Firebird, 2013].

It is important to underline that the experiments were provided for data with fixed length. If keywords’ length is variable, we will have problems to work with any RDBMS. NL-ArM supports variable length of the keywords.

Calculating the hash function is faster than searching keywords in the index. But searching operations in hash structures is slow operation; in this case search in index structures is more convenient.

Another disadvantage of NL-ArM is connected to its special type of realization of the internal index structure. In relational model all keys have same influence on the writing time – they are written in the plain file by the same manner (as parts of records) and extend the balanced index in one or other its section which takes logarithmic time. In NL-ArM the different values of co-ordinates cause various archive structures which take corresponded time. Practically, NL-ArM creates hyper-matrix and large empty zones need additional resources – time and disk space, which are not so great due to smart internal index organization but really exists.

Experiments have shown that regarding NL-ArM:

- In writing, Firebird is on average **90.1 times slower**. This is due to two reasons:
  1. Balanced indexes of Firebird need reconstruction for including of every new keyword. This is time consuming process;
  2. The speed of updating NL-ArM hash tables which do not need recompilation after including new information.

  Due to specific of realization, for small values of co-ordinates NL-ArM is not effective as for the great ones. Nevertheless, NL-ArM is always many times faster than Firebird.
- In reading, Firebird is on average **29.8 times slower** due to avoiding search operations in NL-ArM hash tables which speeds the access.

Finally, for large dynamic data sets more convenient are hash based tools like NL-ArM because of random direct access to all stored records immediately after writing it without any additional indexing. For storing big semi-structured datasets like large ontologies and RDF-graphs this advantage is crucial.

7.2 Analysis of experiments with structured datasets

In Chapter 5 we have presented several experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of structured datasets.

We have provided three main experiments - for NL-storing of dictionaries, thesauruses, and ontologies.

Analyzing results from the experiment with a real dictionary data we may conclude that it is possible to use NL-addressing for storing such information.

Next experiment was aimed to answer to question: "What we gain and loss using NL-Addressing for storing thesauruses?"

Analyzing results from the experiment we point that the loss is additional memory for storing hash structures which serve NL-addressing. But the same if no great losses we will have if we will build balanced search trees or other kind in external indexing. It is difficult to compare with other systems because such information practically is not published. The benefit is in two main achievements:

- High speed for storing and accessing the information;
- The possibility to access the information immediately after storing without recompilation the database and rebuilding the indexes.

The third experiment considered the complex graph structures such as ontologies. The presented survey of the state of the art in this area has shown that main models for storing ontologies are sequential files and relational databases (i.e. sets of interconnected indexed sequential files with fixed records' structure).

Our experiment confirmed the conclusion about losses and benefits from using NL-addressing given above for thesauruses. The same is valid for more complex structures.

For static structured datasets it is more convenient to use standard utilities and complicated indexes. NL-addressing is suitable for dynamic processes of creating and further development of structured datasets due to avoiding recompilation of the database index structures and high speed access to every data element.

7.3 Analysis of experiments with semi-structured datasets

In Chapter 6 the applicability of NL-addressing for middle-size and large semi-structured RDF-datasets was concerned.

We have provided twelve experiments with middle-size and large RDF-datasets, based on selected datasets from DBpedia's homepages and Berlin SPARQL Bench Mark (BSBM) to make comparison with published benchmarks of known RDF triple stores.

➢ ***Rank-based multiple comparison***

We will use the Friedman test to detect statistically significant differences between the systems [Friedman, 1940]. The Friedman test is a non-parametric test, based on the ranking of the systems on each dataset. It is equivalent of the repeated-measures ANOVA [Fisher, 1973]. We will use Average Ranks ranking method, which is a simple ranking method, inspired by Friedman's statistic [Neave & Worthington, 1992]. For each dataset the systems are ordered according to the time measures and are assigned ranks accordingly. The best system receives rank 1, the second – 2, etc. If two or more systems have equal value, they receive equal rank which is mean of the virtual positions that had to receive such number of systems if they were ordered consecutively each by other.

Let *n* is the number of observed datasets; *k* is the number of systems.

Let i_{rj} be the rank of system *j* on dataset *i*. The average rank for each system is calculated as

$$R_j = \frac{1}{n}\sum_{i=1}^{k} r_j^i \,.$$

The null-hypothesis states that if all the systems are equivalent than their ranks R_j should be equal. When null-hypothesis is rejected, we can proceed with the Nemenyi test [Nemenyi, 1963] which is used when all systems are compared to each other. The performance of two systems is significantly different if the corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$. Some of the values of q_α are given in Table 59 [Demsar, 2006].

Table 59. Critical values for the two-tailed Nemenyi test

systems	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\mathbf{q_{0.05}}$	1.960	2.343	2.569	2.728	2.850	2.949	3.031	3.102	3.164
$\mathbf{q_{0.10}}$	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.855	2.920

The results of the Nemenyi test are shown by means of critical difference diagrams.

Experiments which we will take in account were presented in corresponded tables of Chapter 6 as follow (Table 60):

Table 60. Information about tests and results

test No:	**results**
1	Table 40
2	Table 42
3	Table 44
4	Table 46
5a	Table 48 (a)
5b	Table 48 (b)
6	Table 50
7	Table 52
8	Table 54
9	Table 56
10a	Table 57 (a)
10b	Table 57 (b)

Benchmark values from our 12 experiments and corresponded published experimental data from BSBM team are given in Table 61. Published results do not cover all table, i.e. we have no values for some cells. To solve this problem we will take in account only the best result for given system on concrete datasets (Table 62). Sesame had no average values for tests 10a and 10b. Because of this we will not use these test in our comparison. They were useful to see the need of further refinement of RDFArM for big data.

The ranks of the systems for the ten tests are presented below in Table 63.

Table 61. Benchmark values for middle size datasets

system	**TEST**											
	1	**2**	**3**	**4**	**5a**	**5b**	**6**	**7**	**8**	**9**	**10a**	**10b**
RDFArM	3	2272	14.79	3469	60	60	301	136412	1453	5901	15742	31484
Sesame Native (spoc, posc)	3	2404	19	2341	179	213	1988	21896	44225	282455		
Virtuoso (ogps, pogs, psog, sopg)	2	1327		1235			609	7017			6566	14378
Virtuoso TS			05		23	25			2364	28607		
Virtuoso RDF views			09									
Virtuoso SQL			09		34	33			1035	3833		
Virtuoso RV					34	33			1035	3833		
Jena SDB	5		13		129		1053		14678	139988		
Jena TDB					49	41			1013	5654	4488	9913
Jena SDB MySQL Layout 2 Index		5245		6290				70851				
Jena SDB Postgre SQL Layout 2 Hash		3557		3305				73199				
Jena SDB Postgre SQL Layout 2 Index		9681		9640				734285				

Table 62. Chosen benchmark values for middle size datasets

system	TEST											
	1	**2**	**3**	**4**	**5a**	**5b**	**6**	**7**	**8**	**9**	**10a**	**10b**
RDFArM	3	2272	14.79	3469	60	60	301	136412	1453	5901	15742	31484
Sesame	3	2404	19	2341	179	213	1988	21896	44225	282455		
Virtuoso	2	1327	05	1235	23	25	609	7017	1035	3833	6566	14378
Jena	5	3557	13	3305	49	41	1053	70851	1013	5654	4488	9913

Table 63. Ranking of tested systems

system	ranks for the tests										average rank
	1	**2**	**3**	**4**	**5a**	**5b**	**6**	**7**	**8**	**9**	
RDFArM	2.5	2	3	4	3	3	1	4	3	3	**2.85**
Sesame	2.5	3	4	2	4	4	4	2	4	4	**3.35**
Virtuoso	1	1	1	1	1	1	2	1	2	1	**1.2**
Jena	4	4	2	3	2	2	3	3	1	2	**2.6**

All average ranks are different. The null-hypothesis is rejected and we can proceed with the Nemenyi test. Following [Demsar, 2006], we may compute the critical difference by formula:

$$CD = q_{\alpha}\sqrt{\frac{k(k+1)}{6N}}$$

where q_{α} we take as $q_{0.10}$ = 2.291 (from Table 59 [Demsar, 2006; Table 5a]); *k* will be the number of systems compared, i.e. *k*=4; N will be the number of datasets used in benchmarks, i.e. N=10. This way we have:

$$CD_{0.10} = 2.291 * \sqrt{\frac{4*5}{6*10}} = 2.291 * \sqrt{\frac{20}{60}} = 2.291 * 0.577 = 1.322$$

We will use for critical difference $CD_{0.10}$ the value 1.322.

At the end, average ranks of the systems and distance to average rank of the first one are shown in Table 64.

Table 64. Average ranks of systems and distance to average rank of the first one

place	system	average rank	Distance between average rank of the system and average rank of the first one
1	**Virtuoso**	**1.2**	**0**
2	**Jena**	**2.6**	**1.4**
3	**RDFArM**	**2.85**	**1.65**
4	**Sesame**	**3.35**	**2.15**

The visualization of Nemenyi test results for tested systems is shown on Figure 71.

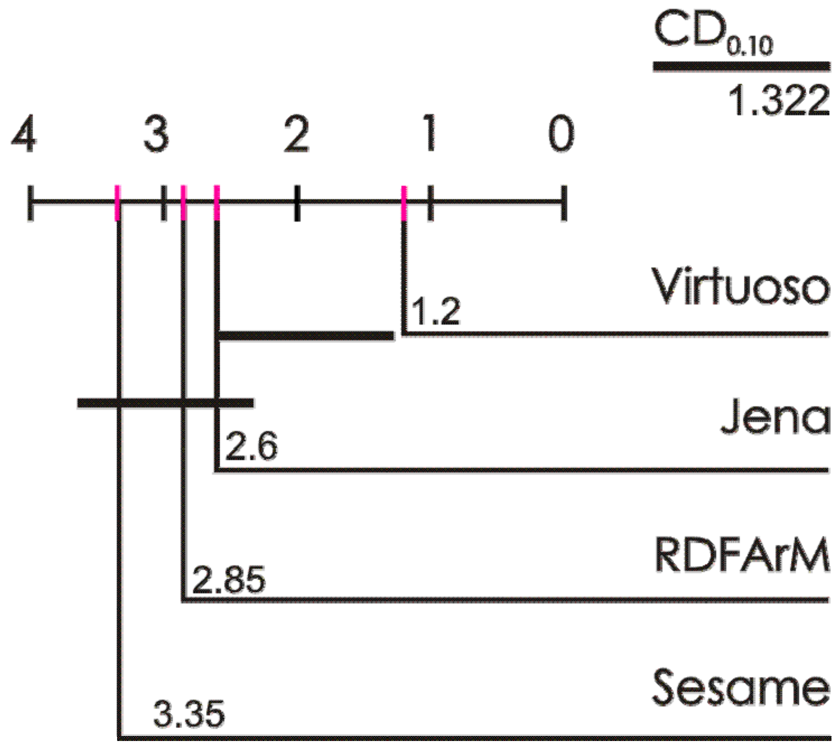


Figure 71. Visualization of Nemenyi test results

Analyzing these experiments we may conclude that RDFArM is at critical distances to Jena and Sesame. RDFArM is nearer to Jena than to Sesame. RDFArM, Jena, and Sesame are significantly different from Virtuoso.

Some recommendations to RDFArM may be given. RDF triple datasets has different characteristics depending of their origination. This causes the need to adapt NL-ArM storage engine to specifics of concrete datasets. For instance, important parameters are length of strings and quantity of repeating values of subject, relation, and object.

7.4 Storing time and multi-processing

The NL-ArM characteristic, which we will analyze now, is NL-access time dependence on growing of dataset size and possibility for multi-processing. Below we will outline data for NL-storing instances with one-, two- and three-elements. Graphical illustrations will be given for loading selected datasets. For two datasets will be given graphical comparison between times consumed by different processors.

✓ *NL-storing two-element instances*

Two-element instances we use in experiments for NL-storing dictionaries (Table 25) and thesauruses (Table 27). Two-element instance is couple: *(name, value)*, where "name" is "one-dimension" co-ordinate of NL-location where "value" (a string) has to be stored.

Measured times are gathered in Table 65.

Table 65. Access times for two-element instances

dataset	number of instances	time for all instances in ms	time for one instance in ms
NL-writing of one-element instances			
SA dictionary	23412	22105	0.94
WordNet as thesaurus	125062	107157	0.86
Total:	**148474**	**129262**	**0.87**
NL-reading of one-element instances			
SA dictionary	23412	20826	0.89
WordNet thesaurus	117641	91339	0.78
Total:	**141053**	**112165**	**0.79**

The average times are **0.87 ms** for writing and **0.79 ms** for reading.

➢ *NL-storing instances with three elements (RDF-triple datasets)*

Three-element instances we use in experiments for NL-storing RDF-triple datasets. Three-element instance is triple: *(subject, relation, object)*, where "subject" and "relation" are NL-locations, and "object" is a string to be stored at NL-location given by two-dimensional co-ordinates *(subject, relation)* where we store only one value (object) which is a string.

Measured times are gathered in Table 66.

Table 66. Loading times for three-element instances

dataset	number of triples	loading time in ms	
		for all triples	for one triple
BSBM 50K	50116	112851	2.3
homepages-fixed.nt	200036	727339	3.6
BSBM 250K	250030	575069	2.3
geocoordinates-fixed.nt	447517	1110415	2.5
BSBM 1M	1000313	2349328	2.3
BSBM 5M	5000453	11704116	2.3
infoboxes-fixed.nt	15472624	43652528	2.8
BSBM 25M	25000244	56488509	2.3
BSBM 100M	100000112	229343807	2.3
total:	**147421445**	**346063962**	**2.34745**

The average loading time is **2.35 ms**.

➢ ***NL-storing four-element instances (RDF-quadruple datasets)***

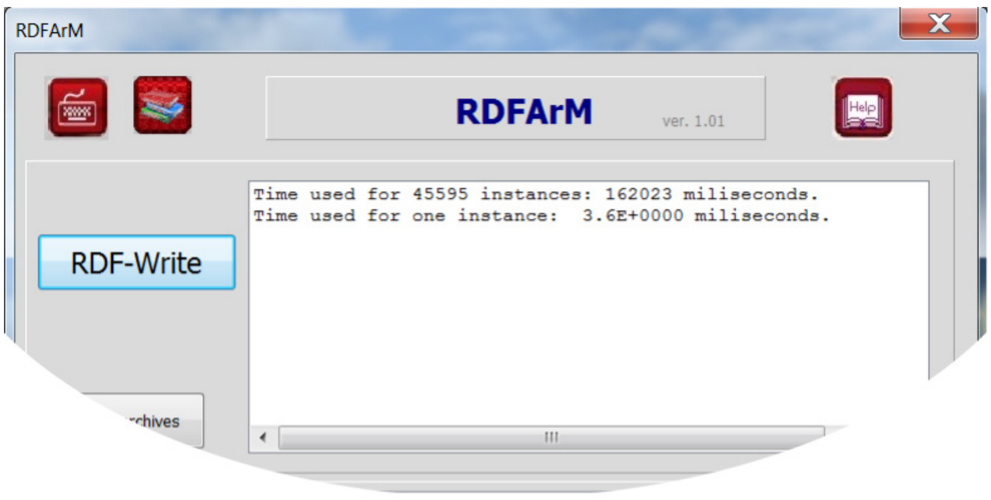
We done series of experiments based on real data from the BTC datasets [BTC, 2012]. Here we will outline only results for dataset http://km.aifb.kit.edu/projects/btc-2012/datahub/data-0.nq.gz.

Dataset "data-0.nq" contains 45595 quadruples. Information about its structure and results from the experiments with it are shown in Table 67. A screenshot from the RDFArM program is shown at Figure 72.

The average loading time is **3.6 ms**.

Table 67. Results for storing datahub/data-0.nq

number of				storing time for all instances in ms	storing time for one instance in ms
instances	subjects	relations	contexts		
45595	7325	894	101	162023	**3.6**



```
Time used for 45595 instances: 162023 miliseconds.
Time used for one instance:  3.6E+0000 miliseconds.
```

Figure 72. A screenshot from the RDFArM program

The time for NL-storing on **Configuration K** is about:

- 0.7 ms for two-element instances;
- 2.35 ms for three-element instances;
- 3.6 ms for four-element instances.

The conclusion is that every new element of the instance takes about one millisecond additional time. This means that ***NL-storing time is depended on the number of elements*** in the instances.

➢ *Graphical illustrations*

Graphical illustrations of loading selected datasets are given below.

Firstly (Figure 73), we show the graphic of time used for storing of one instance from BSBM 250K dataset. At the beginning RDFArM takes more time due to initialization of the hash structures.

The next graphic (Figure 74) illustrates the variation of storing time due to specifics of the dataset, i.e. the size of elements to be used as NL-addresses – as long are the strings of subject and relation, so long time it takes the string of object to be stored in the archive.

Nevertheless, storing time varies between 2.2 and 2.5 milliseconds.

The next two graphics are aimed to illustrate independence from size of the datasets. On Figure 75 the storing times of BSBM 25M dataset is shown, and on Figure 76 storing times of BSBM 100M dataset are illustrated. On both graphics we see the same regularity – constant time for storing of one triple independently of the number of already stored ones.

Simulating multi-processors work we are interested of regularity of used times from different processors. It is expectable to have similar times because of constant complexity of NL-addressing. This is seen of Figure 77 and Figure 78 where graphical comparison between times consumed by different processors is given.

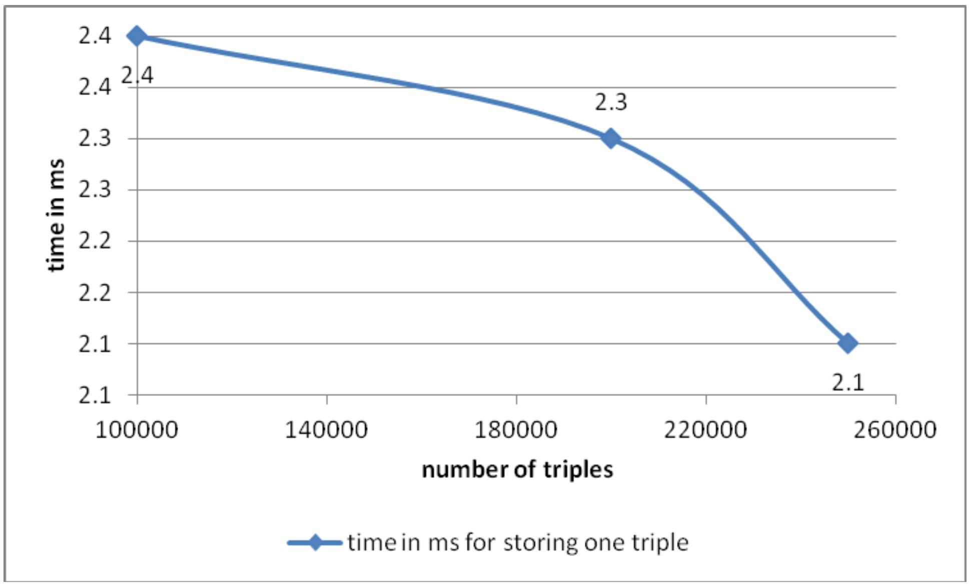


Figure 73. Storing time for one instance of BSBM 250K

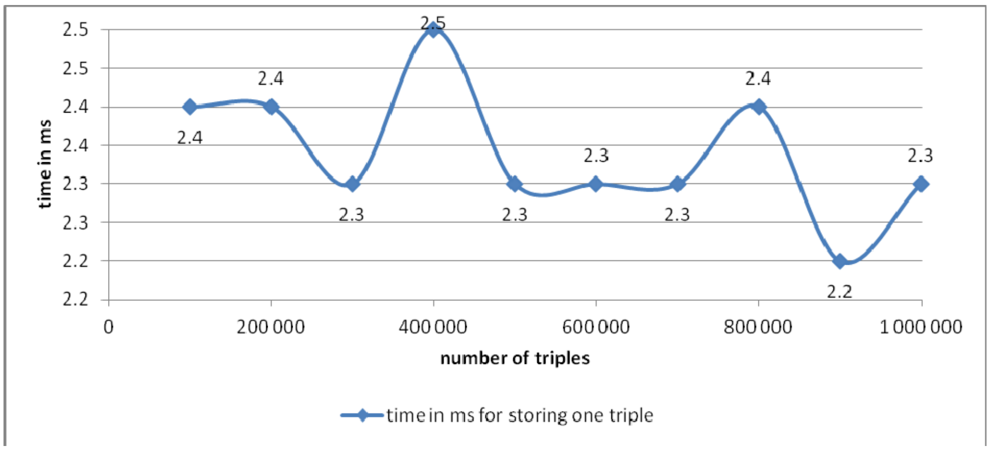


Figure 74. Storing time for one instance of BSBM 1M

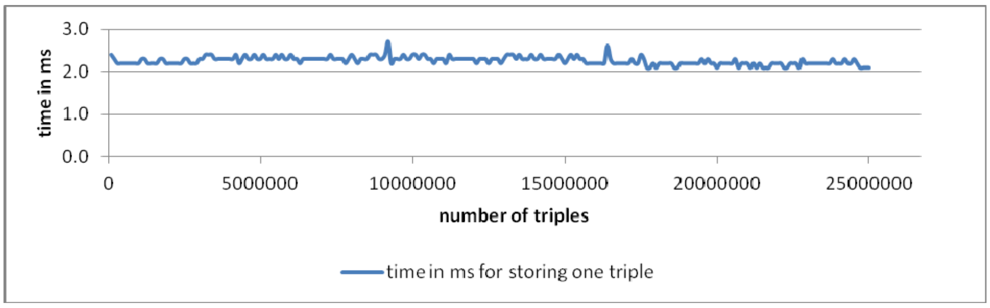


Figure 75. Storing time for one instance of BSBM 25M

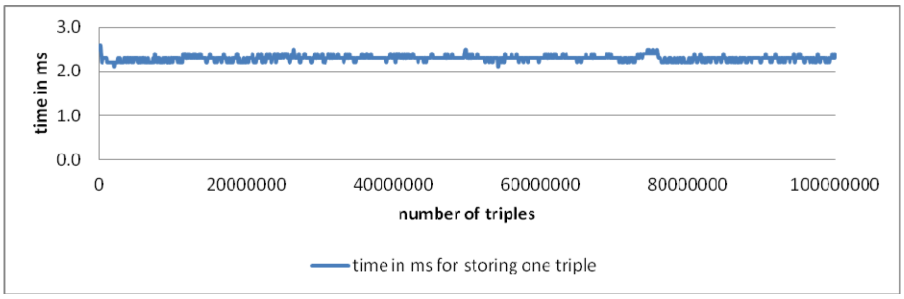


Figure 76. Storing time for one instance of BSBM 100M

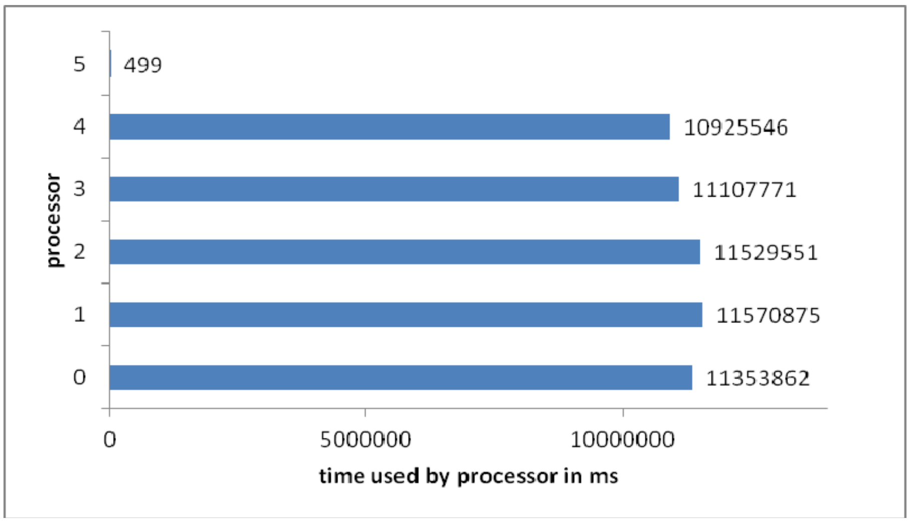


Figure 77. Comparison of time used by processors for BSBM 25M

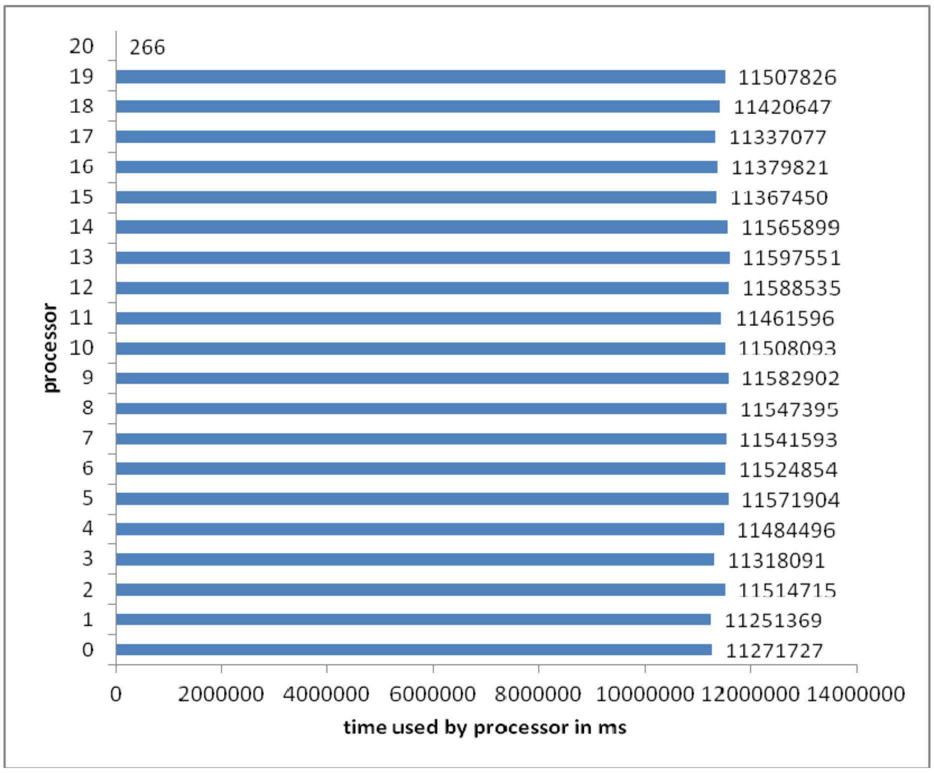


Figure 78. Comparison of time used by processors for BSBM 100M

From experimental data and visualizations we may conclude that the NL-access time:

- ***Depends on number of elements in a dataset's instances***, which have to be stored on the disk;
- ***Not depends on number of instances*** in the dataset.

The second is very important for multi-processing because it means linear reverse dependence on number of processors.

This time does not depend on the size of data sets, i.e. of the number of instances. Detailed information for storing 100 millions triples is given in the Appendix A.5 (Table 71). Table 71 contains results from an experiment for storing 100 millions triples from BSBM 100M [BSBMv3, 2009]. The check points were on every 100 000 triples. For every check point in Table 71, the average time in ms for writing one triple is shown. For comparison, the corresponded value of log n is given in third column. Data from Table 71 are visualized in Figure 79.

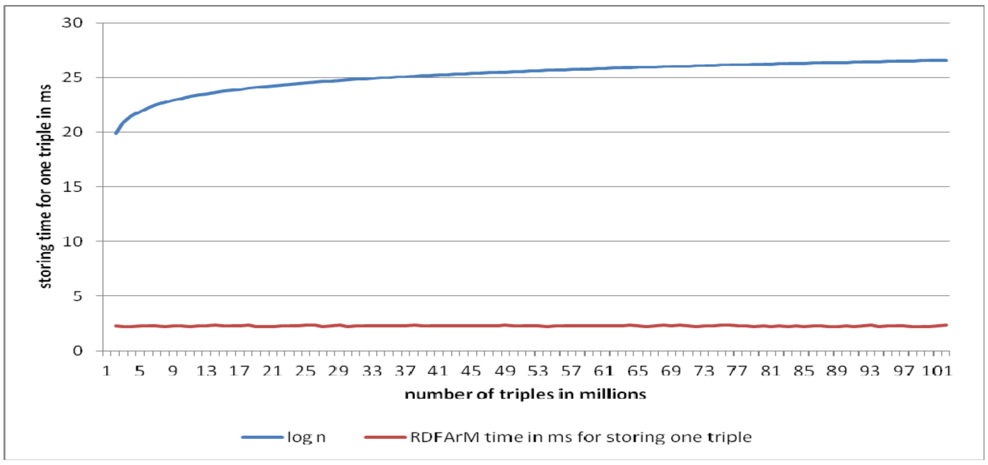


Figure 79. Comparison of log n and average time in ms for storing one triple from BSBM 100M

➢ *Conclusion of chapter 7*

In this chapter we have analyzed experiments presented in previous chapters 4, 5, and 6, which contain respectively results from (1) basic experiments; (2) experiments with structured datasets; (3) experiments with semi-structured datasets. Special attention was paid to analyzing of storing times of NL-ArM access method and its possibilities for multi-processing.

From experimental data and visualizations we concluded that the NL-access time:

— *Depends on number of elements in a dataset's instances, which have to be stored on the disk;*

— *Not depends on number of instances in the dataset.*

The second is very important for multi-processing because it means linear reverse dependence on number of processors.

In Appendix B we outlined some systems which we have analyzed in accordance of further development and implementing of NL-addressing. Two main groups of systems we have selected are:

- *DBMS based approaches (non-native RDF data storage): Oracle [Oracle, 2013], 3Store [AKT Project, 2013], Jena [Jena, 2013], RDF Suite [RDF Suite, 2013], Sesame [Sesame, 2012], 4store [4store, 2013];*
- *Multiple indexing frameworks (native RDF data storage): YARS [YARS, 2013], Kowari [Kowari, 2004], Virtuoso [Virtuoso, 2013], RDF-3X [Neumann & Weikum, 2008], Hexastore [Weiss et al, 2008], RDFCube [Matono et al, 2007], BitMat [Atre et al, 2009], Parliament [Kolas et al, 2009].*

Taking in account our experiments with relational data base we may conclude that for group of DBMS based approaches we will have similar proportions if we realize NL-addressing for more qualitative hardware platforms, for instance cluster machines.

Our approach is analogous to multiple indexing frameworks. The main difference is in reducing the information via NL-addressing and avoiding its duplicating in indexes. Again, if we realize NL-addressing for more qualitative hardware platforms, we will receive results which will outperform the analyzed systems.

What gain and loss using NL-Addressing for RDF storing?

The loss is additional memory for storing internal hash structures. But the same if no great losses we will have if we will build balanced search trees or other kind in external indexing. It is difficult to compare with other systems because such information practically is not published.

The benefit is in two main achievements:

- *High speed for storing and accessing the information;*
- *The possibility to update and access the information immediately after storing without recompilation the database and rebuilding the indexes. This is very important because half or analyzed systems do not support updates (see Table 77).*

The main conclusion is optimistic. The future realization of NL-addressing for cluster machines and corresponded operation systems is well-founded.