Conclusion

The main goal of this research was to study a new approach for storing semi-structured datasets. To achieve this goal, we have studied and analyzed the existing methods and systems for storing semi-structured datasets and we have proposed an information model for storing semi-structured datasets and corresponded access method as well as tools for working in such style, theirs main principles, and storing functions.

We have provided experiments and practical approbation of the proposed model and tools by experimental software realizations and comparative evaluating in order to study their behavior under practical conditions and to compare with other tools from the same class. The main conclusion is optimistic. The future realization of NL-addressing, for instance – for cluster machines and corresponded operation systems, is well-founded.

Our further research will be directed to several interesting areas of implementing the NL-addressing in business applications where flexibility of this approach will give some new possibilities. Implementing the NL-addressing in linguistic systems which work with large linguistic data sets is another direction for further work.

Let point the area of cognitive modeling, too. It is clear; the human brain does not create indexes. The information processing in the brain looks like our model for NL-addressing. It is very interesting to provide research in this area.

Big Data

Maybe the most interesting is the area of so called “Big Data”. The term Big Data applies to information that can’t be processed or analyzed using traditional processes or tools. Increasingly, organizations today are facing more and more “Big Data challenges”. They have access to a wealth of information, but they don’t know how to get value out of it because it is sitting in its most raw form or in a semi-structured or unstructured format [Zikopoulos et al, 2012].

Popular approach for representing Big Data is Resource Definition Framework (RDF). Let remember, RDF is a graph based data format which is schema-less, thus unstructured, and self-describing, meaning that graph labels within the graph describe the data itself. The prevalence of RDF data is due to variety of underlying graph based models, i.e. almost any type of data can be expressed in this format including relational and XML data [Faye et al, 2012].

Big Data created the need for a new class of capabilities to augment the way things are done today to provide better line of site and controls over our existing knowledge domains and the ability to act on them.

BigArM

In the Big Data community, the "MapReduce Paradigm" has been seen as one of the key enabling approaches for meeting the continuously increasing demands on computing resources imposed by massive data sets. MapReduce is a highly scalable programming paradigm capable of processing massive volumes of data by means of parallel execution on a large number of commodity computing nodes. It was recently popularized by Google [Dean & Ghemawat, 2008], but today the MapReduce paradigm has been implemented in many open source projects, the most prominent being the Apache Hadoop [Hadoop, 2014]. The popularity of MapReduce can be accredited to its high scalability, fault-tolerance, simplicity and independence from the programming language or the data storage system.

At the same time, MapReduce faces a number of obstacles when dealing with Big Data including the lack of a high-level language such as SQL, challenges in implementing iterative algorithms, support for iterative ad-hoc data exploration, and stream processing [Grolinger et al, 2014].

A possible solution may be the approach of Natural Language Addressing (NLA) presented in this monograph. It is suitable for storing Big Data. Its main idea is to use internal encoding of letters of a word or phrase as elements of co-ordinate vector which may be used as hyper-space address of the information connected to this word or phrase. As result the standard indexing and recompilation of information base are avoided.

Three main characteristics define Big Data: ***Volume, Variety, and Velocity*** [Zikopoulos et al, 2012]. These characteristics cause corresponded problems of storing Big Data which may be solved by means of NLA [Markov et al, 2014]:

- **Volume** (the sheer volume of data being stored today is exploding) – avoiding additional indexing, duplication of keywords, and corresponded pointers, leads to reducing additional memory needed for accessing information i.e. we may use addressing but not classical search engines;
- **Velocity** (a conventional understanding of velocity typically considers how quickly the data is arriving and stored, and its associated rates of retrieval) – avoiding recompilation of information base permits high speed of storing and immediately readiness of information to be accessed. This is very important possibility for stream data;
- **Variety** (it represents all types of data — a fundamental shift in analysis requirements from traditional structured data to include raw, semi-structured, and unstructured data as part of the decision-making and insight process) – natural language addressing permits creating a special kind of graph information bases which may operate both with structured as well as semi-structured information.

What is needed is to extend possibilities of ArM32 up to 64 bit addressing capabilities and to rationalize the internal hash structures to speed access from milliseconds down to microseconds per one access operation. This will be done in ongoing developing of its new version called "BigArM" for 64 bit machines and operating systems like MS Windows and Linux as well as for Cloud processing.

Collect/Report Paradigm

Realizing BigArM will permit new kind of Cloud processing of Big Data, called "Collect/Report Paradigm" (CRP). Its idea is very simple and because of this it is perspective to be realized.

CRP is based on the possibility of NLA to separate incoming information coded as RDF-triples on many different layers stored in separate archives which may be distributed all over the world. The correspondence between archives is strongly kept by names as addresses which are equal for all layers.

Similar model we may see in the game of chance "Bingo" (Figure 84) for two or more players, who mark off numbers on a grid with unique sequence of numbers printed on their individual cards as they are announced by the Caller corresponding to numbered balls drawn at random; the game is won by the first person to call out "bingo!" or "house!" after crossing off all numbers on the grid or in one line of the grid [YourDictionary, 2013].

Figure 84. Illustration of Collect/Report Paradigm via example of Bingo game

To play Bingo one has to "***collect***" (to buy) one or more individual cards and after starting the game to listen what number the Caller will announce, to find in the individual cards the same numbers and to mark them (i.e. to process the stream of incoming data). After marking every new number, (in real time, before next number will be announced) player has to analyze the configuration of marked cells on the individual cards and to decide if it is the winner configuration. If the configuration is a winner one, the player has to "***report***" (to call out) "Bingo". Only the players with winner configurations have to report, the others must stay silent.

In Collect/Report Paradigm, all nodes have to "listen" in parallel the incoming stream of RDF-data and to "collect" (to store) information only in the layers the nodes have to support. In the same time, nodes have to "listen" incoming stream of requests and only nodes, which have information corresponded to given request has to "report" (to send answer).

As an example let's remember a part from Table 32 (Table 69). Let it represents six nodes numbered from 1 to 6 which may be distributed over the net. Incoming information is in RDF triples (subject, relation, object). Information (objects) for the same subject and relation is concatenated in the corresponded points. Let assume that Table 69 represents the state of nodes at given time moment. If in this moment a request for word "cut" will come, only nodes 1 and 6 will "report" the content (definitions) from corresponded cells. Node 1 will report only the first row which correspond to "cut" with small letters but not its second row which corresponds to word "CUT" with capital letters. Nodes 2, 3, 4, and 5 will rest silent.

Table 69. A part from Table 32

node	layer	NLA	definition
1	**adj_all**	cut	{ cut, shortened, (with parts removed; "the drastically cut film") } { cut, thinned, weakened, (mixed with water; "sold cut whiskey"; "a cup of thinned soup") } { cut, slashed, ((used of rates or prices) reduced usually sharply; "the slashed prices attracted buyers") } { cut, emasculated, gelded, ((of a male animal) having the testicles removed; "a cut horse") }
		CUT	{ [CUT1, UNCUT1,!] (separated into parts or laid open or penetrated with a sharp edge or instrument; "the cut surface was mottled"; "cut tobacco"; "blood from his cut forehead"; "bandages on her cut wrists") } { [CUT2, UNCUT2,!] ((of pages of a book) having the folds of the leaves trimmed or slit; "the cut pages of the book") }

node	layer	NLA	definition
			{ [CUT3, UNCUT3,!] (fashioned or shaped by cutting; "a well-cut suit"; "cut diamonds"; "cut velvet") }
2	**adj_pert**	cut	empty definition
3	**adj_ppl**	cut	empty definition
4	**adv_all**	cut	empty definition
5	**noun_Tops**	cut	empty definition
6	**noun_act**	cut	{ cut6, absence,@ (an unexcused absence from class; "he was punished for taking too many cuts in his math class") } { cut5, reduction,@ (the act of reducing the amount or number; "the mayor proposed extensive cuts in the city budget") } { cut, [cutting, verb.creation:cut11,+] cutting_off1, shortening,@ (the act of shortening something by chopping off the ends; "the barber gave him a good cut") } { cut1, [cutting1, verb.contact:cut10,+ verb.contact:cut,+] division,@ (the act of cutting something into parts; "his cuts were skillful"; "his cutting of the cake made a terrible mess") } { cut2, [cutting2, verb.contact:cut10,+] opening2,@ (the act of penetrating or opening open with a sharp edge; "his cut in the lining revealed the hidden jewels") } { cut9, [cutting9, verb.contact:cut5,+] division,@ card_game,#p (the division of a deck of cards before dealing; "he insisted that we give him the last cut before every deal"; "the cutting of the cards soon became a ritual") } { cut8, [undercut, verb.contact:undercut,+] stroke,@ tennis,;c badminton,;c squash,;c ((sports) a stroke that puts reverse spin on the ball; "cuts do not bother a good tennis player") }

In general, Collect/Report Paradigm is illustrated on Figure 85.

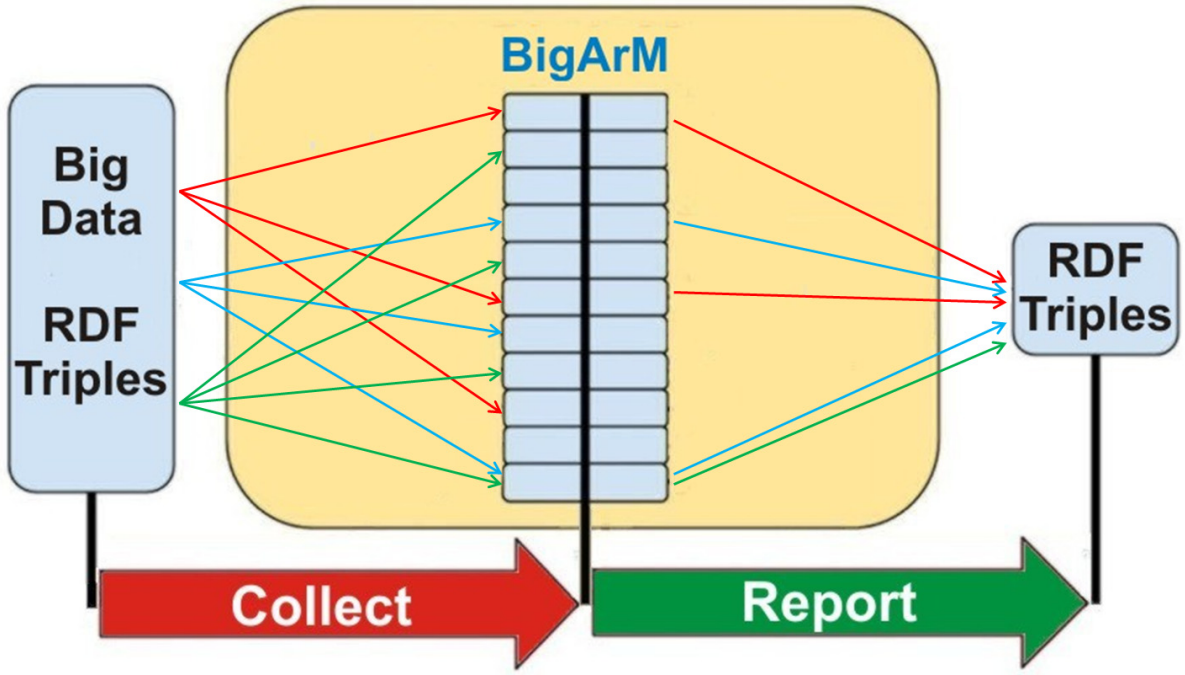

Figure 85. Cloud Collect/Report Scheme for Storing and Accessing Big Data

Main advantages of Collect/Report Paradigm (Figure 85) are:

- Collecting information is done by all nodes independently in parallel. It is possible one node to send information to another;
- Reporting information is provided only by the nodes which really contain information related to the request; the rest nodes do not react, they remain silent;
- Input data as well as results are in RDF-triple or RDF-quadruple format.

Main results presented in the monograph

Chapter *1 introduced the main data structures and storing technologies which further we will use to compare our results. Mainly they are graph data models as well as RDF storage and retrieval technologies.*

Firstly we defined concepts of storage model and data model.

Mapping of the data models to storage models is based on program tools called "access methods". Their main characteristics were outlined.

Graph models and databases were discussed more deeply and examples of different graph database models were presented. The need to manage information with graph-like nature especially in RDF-databases had reestablished the relevance of this area.

There is a real need of efficient tools for storing and querying knowledge using the ontologies and the related resources. In this context, the annotation of unstructured data has become a necessity in order to increase the efficiency of query processing. Efficient data storage and query processing that can scale to large amounts of possibly schema-less data has become an important research topic. The proposed approaches usually rely on (object-) relational database technology or on main-memory virtual machine implementations, while employing a variety of storage schemes [Faye et al, 2012].

In accordance with this, the analyses of RDF databases as well as of the storage and retrieval technologies for RDF structures were in the center of our attention. The analysis of the viewed tools showed that all of them use data storing models which are limited to text files, indexed data or relational databases. These approaches do not conform to the specific structures of the ontologies. This necessitates the development of new models and tools for storing ontologies which correspond to their structure.

Storing models for several popular ontologies and summary of main types of storing models for ontologies and, in particular, RDF data were discussed.

At the end of this chapter, our attention was paid to addressing and naming (labeling) in graphs with regards to introducing the Natural Language Addressing (NL-addressing) in graphs. A sample graph was analyzed to find its proper representation.

Taking in account the interrelations between nodes and edges, we saw that a "multi-layer" representation is possible and the identifiers of nodes and edges can be avoided.

Concluding, let us point on advantages and disadvantages of the multi-layer representation of graphs.

The main disadvantages are:

- *The layers are sparsed;*
- *The number of locations may be very great which causes the need of corresponded number of columns in the table (in any cases hundred or thousand).*

The main advantages are:

- *Reducing the used resources;*
- *The NL-addressing means direct access to content of each cell. Because of this, for NL-addressing the problem of recompiling the database after updates does not exist. In addition, the multi-layer representation and natural language addressing reduce resources and avoid using of supporting indexes for information retrieval services (B-trees, hash tables, etc.);*
- *Finally, using NL-addressing, the multi-layer representation is easily understandable by humans and interpretable by the computers.*

If we will use indexed files or relational data bases, the disadvantages are so serious that make the implementation impossible.

We propose to use a multi-dimensional model for organization of information. It is presented in next chapter.

***Chapter 2** aimed to introduce the theoretical surroundings of our work.*

Firstly in this chapter, we remembered the needed basic mathematical concepts. Special attention was paid to the Names Sets – mathematical structure which we implemented in our research. We used strong hierarchies of named sets to create a specialized mathematical model for new kind of organization of information bases called "Multi-Domain Information Model" (MDIM). The "information spaces" defined in the model are kind of strong hierarchies of enumerations (named sets).

*We will realize MDIM via special kind of hashing. Because of this, we remembered the main features of hashing and types of hash tables as well as the idea of "Dynamic perfect hashing" and "Trie", especially – the "Burst trie". A **burst trie** is an in-memory data structure, designed for sets of records that each has a unique string that identifies the record and acts as a key. Burst trie consists of three distinct components: a set of records, a set of containers, and an access trie.*

***Chapter 3** was aimed to introduce a new access method based on the idea of Natural Language Addressing.*

*MDIM and its realizations are not ready to support NL-addressing. We upgraded them for ensuring the features of NL-addressing via new access method called **NL-ArM**.*

The program realization of NL-ArM is based on specialized hash functions and two main functions for supporting the NL-addressing access.

In addition, several operations were realized to serve the work with thesauruses and ontologies as well as work with graphs.

*NL-ArM is ready for storing RDF information. It is possible to define tree information models for storing RDF-graphs using NL-ArM: (1) **RSO model** (Relation-Subject-Object model), (2) **SRO model** (Subject-Relation-Object model), and (3) **UNL model** (**(Subject, Relation) => Object** Universal model) .*

*In **Chapter 4** two main types of basic experiments were presented. NL-ArM has been compared with (1) sequential text file of records and (2) relational database 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 semi-structured data. 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.

For 8 characters as length of the keywords and small quantity of records, the NL-ArM archive occupies more memory than text file but for the case of very large data the NL-ArM archive is smaller. It is important to underline that these experiments were based on artificial data with fixed length (record of 30 bytes with 8 bytes artificially generated keyword of arbitrary ASCII symbols). If the length of the keywords is variable, the size of NL-ArM archive will be different according of length of the strings of keywords of stored information, i.e. according of number of layers of hash tables (depth of trie).

In sequential storing of records, NL-ArM access method is slower than same operation in text file. For applications where it is important in real time to register incoming information, the text files are preferable than archives with NL-Addressing.

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.

We did not compare the sizes of files of NL-ArM and Firebird because of difference of keywords – symbols for Firebird and integer values for NL-ArM.

*In writing experiments, regarding NL-ArM, Firebird is on average **90.1 times slower.** This result is due to two reasons. The first is that balanced indexes of Firebird need reconstruction for including of every new keyword. This is time consuming process. The second reason is 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 as effective as for the great ones.*

*In reading experiments, regarding NL-ArM, Firebird is on average **29.8 times slower.** This result is due to the speed of access in NL-ArM hash tables which do not need search operations.*

If we need direct access to large dynamic data sets (via NL-path), than more convenient are hash based tools like NL-ArM. For instance, such cases are large ontologies and RDF-graphs.

*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.*

Firstly we introduced the idea of knowledge representation. Further in the chapter we discussed three main experiments - for NL-storing of dictionaries, thesauruses, and ontologies.

Presentations of every experiment started with introductory part aimed to give working definition and to outline state of the art in storing concrete structures.

The explanation of the experiments begins with the easiest case – storing dictionaries. 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:

(1) High speed for storing and accessing the information.

(2) 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 files and relational databases.

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.

Here we have to note that 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.

The goal of the experiments with different small datasets, like dictionary, thesaurus or ontology, was to discover regularities in the NL-addressing realization. Analyzing Table 25, Table 27, and Table 33 we may see the main two regularities of storing time using NL-addressing:

- *It depends on number of elements in the instances;*
- *It not depends on number of instances in datasets.*

*In **Chapter 6** we have presented results from series of experiments which were needed to estimate the storing time of NL-addressing for middle-size and very large RDF-datasets.*

We described the experimental storing models and special algorithm for NL-storing RDF instances. Estimation of experimental systems was provided to make different configurations comparable. Special proportionality constants for hardware and software were proposed. Using proportionality constants, experiments with middle-size and large datasets become comparable.

Experiments were provided with both real and artificial datasets. Experimental results were systematized in corresponded tables. For easy reading visualization by histograms was given.

The goal experiments for NL-storing of middle-size and large RDF-datasets were to estimate possible further development of NL-ArM. We assumed that its "software growth" will be done in the same grade as one of the known systems like Virtuoso, Jena, and Sesame. In the next chapter we will

analyze what will be the place of NL-ArM in this environment but already we may see that NL-addressing have good performance and NL-ArM has similar results as Jena and Sesame.

*In **Chapter 7** 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.

*In **Chapter 8** some practical aspects of implementation and using of NL-addressing were discussed in this chapter.*

NL-addressing is approach for building a kind of so called "post-relational databases". In accordance with this the transition to non-relational data models was outlined.

The implementations have to be done following corresponded methodologies for building and using of ontologies. Such known methodology was discussed in the chapter. It is called "METHONTOLOGY" and guides in how to carry out the whole ontology development through the specification, the conceptualization, the formalization, the implementation and the maintenance of the ontology.

Special case is creating of ontologies of text documents which are based on domain ontologies. It consists of Document annotation and Ontology population which we illustrated following the known OntoPop platform [Amardeilh, 2006].

The software realized in this research was practically tested as a part of an instrumental system for automated construction of ontologies "ICON" ("Instrumental Complex for Ontology designatioN") which is under development in the Institute of Cybernetics "V.M.Glushkov" of NAS of Ukraine.

In this chapter we briefly presented ICON and its structure. Attention was paid to the storing of internal information resources of ICON realized on the base of NL-addressing and experimental programs WordArM and OntoArM.

ICON is still under developing and, during solving concrete problems, new functions based on NL-addressing and NL-ArM rise to be realized. For instance, such problems concern the operations with ontologies; work with very large ontological structures; etc.

*Finally, in the **Conclusion,** we presented shortly the next steps. Special attention was done on the area of so called "Big Data" and possible implementation of NLA for processing of large semi-structured data sets. New realization of the access method called BigArM and connected to it Collect/Report Paradigm were outlined. Main advantages of Collect/Report Paradigm are (1) Collecting information is done by all nodes independently in parallel. It is possible one node to send information to another; (2) Reporting information is provided only by the nodes which really contain information related to the request; the rest nodes do not react, they remain silent; (3) Results are in RDF-triple or RDF-quadruple format.*