Hyper Nested Graph: Data Model for Big Data

Ali Asghar Safaei[1]

Receive :2016/04/20 Accepted: 2016/07/26

Abstract

in the era of information, data which are worthwhile asset of human, organizations and enterprises have become such sophisticated that the conventional approaches and methods are not usable anymore, or not efficient at least. Such complexity which is known as the Big Data problem is the affordable extraction of value from big data sets that we are encountered in many recent applications *e.g.*, e-business, scientific research, monitoring, search engines, social networking, etc.. Big Data complexities are instantiated by three major dimensions, high *Volume*, high *Variety*, and high *Velocity* (a.k.a. 3Vs). The first and most essential step in data management (also for Big Data management) is designing and employing a proper data model, as the footstone of the other data management activities such as R&D of DB languages, DBMSs, tools, methods, algorithms, etc.. In this paper, a proper data model for Big Data is designed and proposed in which the properties required for Big Data problem (*i.e.*, to be integrated, complete, scalable, flexible, compatible, and efficient) are considered. As a data model, data *representation* is designed and implicit *integrity constraints* are presented for the proposed *HNG* (*Hyper Nested Graph*) data model. Experimental evaluation results show that the proposed data model outperforms other currently used data models such as the *document-based*, *graph document-based*, and *graph*-based data models in terms of response time.

Keywords- data model, Hyper-Nested Graph, big data, NoSQL, conceptual database modeling

[1] Department of BioMedical Informatics, Faculty of Medical Sciences, Tarbiat Modares University, Tehran. Iran.
aa.safaei@modares.ac.ir

1. INTRODUCTION

Since in the current present, the grow rate of information is beyond the Moor rule, additional information has caused many issues and challenges. The main goal is to benefit the potential of the data to obtain the attainable value from knowledge and value hidden in the massive volume of data. The idea of discovering data is known as ***Big Data*** which includes various applications, such as e-business and e-commerce, banking, leadership, monitoring, and scientific research (such as astronomy, medicine, genetics, geography, etc.).

As estimations show, the amount of commercial data in almost all companies through the world grows nearly twice every 1.2 years [1]. The total amount of generated data of 2011 in the world that was 1.8 zettabyte (around 10^{21} byte), which grows around 9 times every 5 years [2]. So, as Gartner Institute has also stated, the problem of Big Data is one of 10 strategic technology domains and scientific areas in the current and future time.

IDC (one of efficient pioneer in Big Data domain) defines the Big Data problem as affordable value extraction of massive and various data, which is captured, discovered and analyzed very rapidly [1]. Resolving Big Data problem which is in fact converting the data to value, includes different stages such as capturing, storage, transmission, management, processing, analyzing and finally visualizing data and the results [3, 4].

Big data is an abstract term that in addition to have a big volume, includes some further characteristics. At the present, although the importance of big data generally has been known, but still there is no standard definition for it. However, the most prevalent one, defines the big data as data set having high *volume*, high *variety*, and high *velocity* (**3Vs**) [3, 4, 50]:

-***Volume*:** high volume of data (in several petabyte or zettabytes)

-***Variety*:** data variety (as type of data, and sources generating data)

-***Velocity*:** high velocity of data (both for input and output data, in other words, high speed data arrival and data processing)

(Of course, some other definitions even add the fourth "V" as '*Value*', '*Veracity*', or '*Variability*'.)

Unfortunately, common systems, devices and technologies lack capability of managing data sets with these characteristics and consequently, the exploitation and gaining value form such data sets, mostly is not achievable. So, the term *Big Data* issues the problem of offering technologies, systems, devices, methods, models and structures, in which we can extract underlying knowledge from such data sets and convert them to value. The most significant challenges to do this, are as follow:

-*Data representation*

Different of datasets have heterogeneous and diverse data type, structure, meaning, organization, granularity and etc.. Data representation tries to handle these diversities, and makes data meaningful and applicable for analyzers and interpreters (either users or applications). So, improper data representation may lead to missing real value of data and appearing problem in efficient retrieval and analyze of data. Data representation must be designed and determined in conceptual, logical and physical levels, in order to be able to capture and manage data, to be applicable thereafter. The most fundamental part of data representation is designing and using the *data model*.

In this paper, an appropriate data model for big data is proposed.

- *Data compression and redundancy reduction*

Generally, data inherently and potentially have redundancy (even if would not repeated explicitly, often are derived from others). Compression and reduction of data redundancy can have an undeniable role in decreasing the costs, and efficient handling of big data.

- *Data lifecycle management*

In traditional data management (the Relational model), essentially, the final snapshot of data (the last value) is stored and used. In contrast, in big data, it is very different. So, managing data lifecycle and respecting data freshness and data quality, along with keeping historical data as well as using technique like summarization, memory management and so on, play significant roles in making a tradeoff between efficiency and data quality in knowledge discovery and value extraction.

-*Mechanisms and algorithms for data processing and analyzing*

Considering big data characteristics, traditional techniques, algorithms and mechanism for data processing and analyzing need to be revised, in order to be able to process fast, high volume and various data in real-time.

-*Scalability and extensibility*

In accordance to high growth rate in scales of the big data problem (e.g., size, speed, variety), scalability and extensibility should be considered in all levels and dimensions of the solutions (from suitable architecture for the system, to employment of appropriate computational models like *Map-Reduce*, and even designing of details such as address space).

-*Visualization*

Regards to special characteristics of big data, visualization is very challenging in representing results of processing and analysis, and requires some proper models, methods, and tools.

Among these challenges which relate to big data handling, data representation (*i.e.*, data modeling) which is the foundation of data management, may be the most important one.

Generally, data modeling can be done in three levels: *conceptual level*, *logical level* and *physical level* (figure 1).

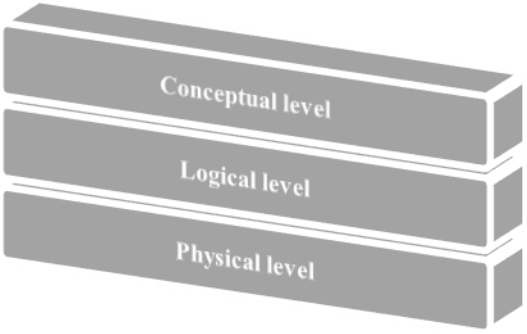


Fig. 1. Levels (layers) of data model

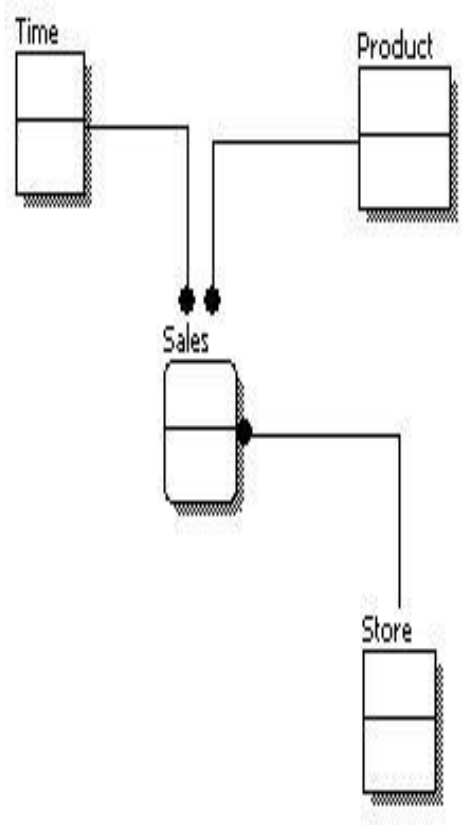
In conceptual level, data domain in related application is determined and represented in an abstract format (including entities as representative agent for real instances, their characteristics, along with the relationships between entities) like *ERD* diagrams and Class diagram in *UML*.

In logical level, independent of the platform in which data are stored and managed on, representation of the entities and relationships

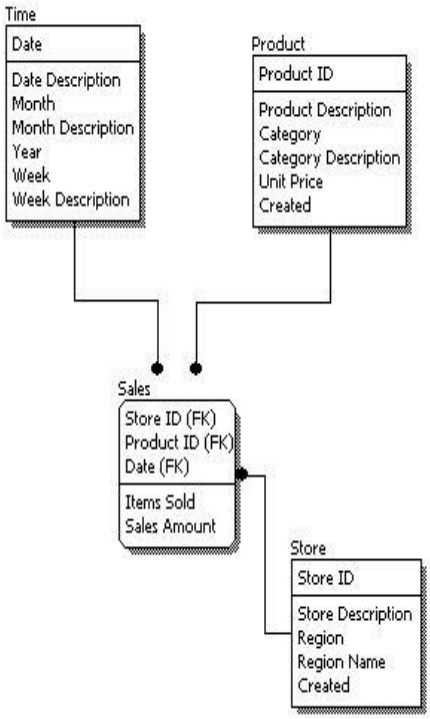
based on a theoretical designed model is provided. The logical model includes *model of representation, set of constraints* on that representation model, and a *set of operators* for manipulation of the data. For example, in the *Relational* model [5] as an example of logical data model, data representation is in the form of *table*, there are some implicit constraints in Relational model such as need for determination of the *key* in every table, and some operators for manipulation of the tables, like what are defined in *Relational Algebra* (R.A.), *e.g., selection, projection, join, etc*.. [6]

In physical level, physical low-level details required for storage and retrieval of data are modeled and determined in accordance to the destination platform's characteristics.

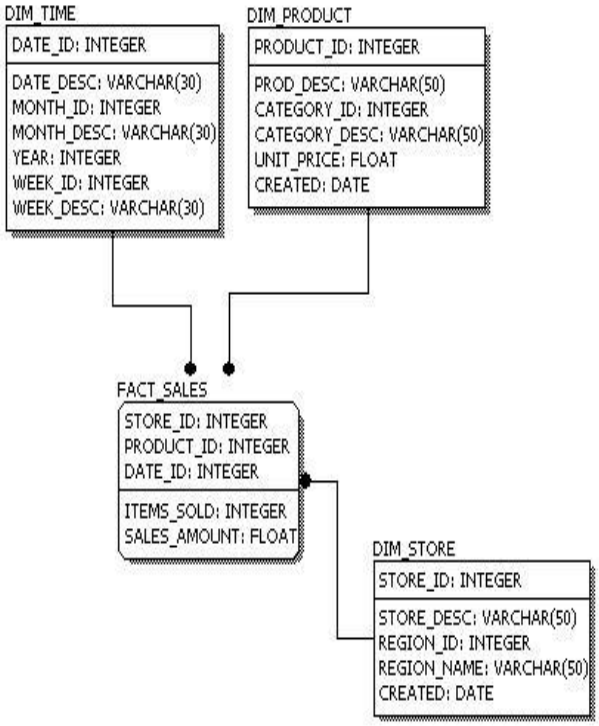
Figure 2 shows an example of data modeling in different levels, for a simple business application.



a) Conceptual Model



b) Logical model



c) Physical model

Fig. 2: data modeling in *(a)* Conceptual *(b)* Logical, and *(c)* Physical levels

The conceptual model, because of its high level of abstraction, does not concern on any specific constraint of the data. So, it does not embrace any essential constraint of big data to model. Physical model, which greatly depends on underlying platform and hardware details, should be designed tightly regarding to the platform.

What should be considered about a new issue in data management (like big data problem) and resolve its challenges in logical level of abstraction is logical data model which is also known as namely as data model (like relational data model, object oriented, object-relational and XML models, or NoSQL data model).

In this study, the appropriate data model for big data problem and the way of addressing to big data requirements and triplet characters (3Vs) have been discussed.

The rest of this study has been structured as follow: related work are discussed in section 2. In section 3, common data model is examined. The proposed data model is described and explained in section 4, and empirically evaluated in section 5. Finally, in section 6 it will be concluded and some of the future work are addressed.

2. RELATED WORK

Data modeling concepts and principles are provided in many chapters of database design books [31-35]. Graph data models are motivated by real-life applications where information about interconnectivity of its pieces is a salient feature. A complete and useful survey of Graph data models concentrating in data structures, query languages, and integrity constraints is presented in [16]. Summarizing, main proposals on Graph data (database) models and their characteristics is provided in table 1 [16].

In [18] a model for management of graph data is proposed to provide analysts with a set of simple, well-defined, and adaptable components to perform complex graph modeling and advanced analysis tasks.

The Hypernode db-model was described in a sequence of papers [36-38]. A hypernode is a directed graph whose nodes can themselves be graphs (or hypernodes), allowing nesting of graphs. Hypernodes can be used to represent simple (flat) and complex objects (hierarchical, composite, and cyclic) as well as mappings and records. A key feature is its inherent ability to encapsulate information.

The Hypernode model was introduced by Levene and Poulovassilis [38], who define the model and a declarative logic-based language structured as a sequence of instructions (hypernode programs), used for querying and updating hypernodes. The implementation of a storage system based on the hypernode model is presented in [39].

In a second version [37], the notion of schema and type checking is introduced via the idea of types (primitive and complex), that are also represented by nested graphs.

The main features of the Hypernode model are: it is based on a nested graph structure which is simple and formal; it has the ability to model arbitrary complex objects in a straightforward manner; it can provide the underlying data structure of an object-oriented data model; it can enhance the usability of a complex objects database system via a graph-based user interface.

As drawbacks, we can mention that data redundancy can be generated by its basic value labels, and that restrictions in the schema level are limited, for example the specification of restrictions for missing information or multivalued relations is not possible.

GROOVY (Graphically Represented Object-Oriented data model with Values [23]) is a proposal of object-oriented db-model which is formalized using hypergraphs, that is, a generalization of graphs where the notion of edge is extended to hyperedge, which relates an arbitrary set of nodes [40]. The model defines a set of structures for an object data model: value schemas, objects over value shemas, value functional dependencies, object schemas, objects over objects schemas and class schemas. The model shows that these structures can be defined in terms of hypergraphs. GROOVY influenced the development of the

Hypernode model providing another approach to modeling complex objects. If we compare both models, we can see that hypergraphs can be modeled by hypernodes by encapsulating the contents of each hyperedge within a further hypernode. In contrast, the multilevel nesting provided by hypernodes cannot easily be captured by hypergraphs [36].

A survey of some graph query languages (including GraphLog, G, GRAM, GraphDB, GooD, G-Log, and GUL) is provided in [41]. A quantitative study and performance comparison of some open-source graph databases is presented in [42]. Current applications and implementations of graph databases, giving an overview of the different types available and their application is studied in [25].

Also, [21] reports on a comparison of one NoSQL graph database called *Neo4j* with a common relational database system, MySQL, for use as the underlying technology in the development of a software system to record and query data provenance information.

Table 1: Main proposals on Graph Database Models and their characteristics ("√" indicates support and "±"} " partial support) [16]
LDM [22], Hypernode [38], GOOD [24], GROOVI [23], GMOD [43], Simatic-XT [44], Gram [45], PaMaL [46], GOAL [47], Hypernode2 [36], Hypernode3 [37], GGL [48], GDM [49].

Characteristics | Database Model		LDM	Hypernode	GOOD	GROOVY	GMOD	Simatic-XT	Gram	Pamal	GOAL	Hypernode2	Hypernode3	GGL	GDM
		1984	1990	1990	1991	1992	1992	1992	1993	1993	1994	1995	1995	2002
Basic Foundation	Graph model	√		√		√		√	√	√			√	√
	Hypergraphs				√									
	Hypernode		√				√				√	√	√	
	Object Oriented model	√		√	√	√			√	√				
Digragh	Node Labeled	√	√	√	√	√	√	√	√	√	√	√	√	√
	Edge Labeled			√	√	√	√	√	√	√			√	√
Support:	Schema	√		√	√	√		√	√	√	√	√		√
	Complex objects	√	√		√		√		√	√	√	√	√	√
	Higher-order relations		√		√					±	√	√	√	√
	Tuples	√							√					√
	Sets	√							√					
	N-ary relations									√				√
	Grouping		√		√		√				√	√	√	
	Derivation and Inheritance				√				√	√	±			√
	Nested relations		√		√		√				√	√	√	
Query Language	Algebraic - Procedural	√					√	√					√	
	Logic - Declarative	√	√		√						√		√	
	Graphical			√		√								
	Query	√	√		√		√	√	√		√	√	√	
	Trans formation	√		√		√				√				√
	Path queries		±	±			√	√	±	±	±	±	±	
Integrity Constraints	Schema-Instance Consistency	√		√	√	√		√	√	√	√	√		√
	Identity and Referential Integrity		√	±		±				±				±
	Functional Dependencies				√							√		
Implementation			√	√			√	√			√	√	√	√
Motivation		Complex objects	Complex objects	Graphical Interfaces	General	Hyper media	Tramport networks	Hypertext	Graphical Interfaces	Graphical Interfaces	Complex objects	Hypertext	Genomics	Complex objects

3. BACKGROUND: CURRENTLY USED DATA MODELS

Various logical data models have been introduced and used during these years; from obsoleted *Hierarchical* [7] and *Network* [8] models (which emphasize the physical level, and offer the user the means to navigate the database at the record level, thus providing low level operations to derive more abstract structures), or special purpose models such as the *Functional* or *Deductive* models, to the traditional *Relational* data models which has been introduced in 1969 by Edgar F. Codd [5, 9] and is still used in many classic business applications.

Popularity and perdurability of the Relational data model has some important reasons as follows:

+ It is very simple and understandable for everyone

It is based on concept of the relation in set theory and nearby everyone knows *rows* and *columns* of a *table*

+ Powerful theoretical background

Theoretical tools such as the *R.A.* (Relational Algebra), *D.R.C.* (Domain Relational Calculus), and *T.R.C.* (Tuple Relational Calculus) and also complete and powerful software tools such as DBMSs (*e.g.*, Oracle, MS SQL Server, IBM DB2, PostgreSQL, My SQL, Informix, etc.).

+ Pervasive research activities

A vast amount of researches in all fields related to data management using the Relational data model had been performed that have caused many of these aspects (*e.g.*, normalization, query, optimization, integrity, security and etc.) to become a routine process. But, despite these advantages, the relational data model has some essential defects.

- Improper performability in some new applications such as CAD, CAM, etc.

- Insufficient support of new data types e.g., voice, image, etc.

- Lack of support for nested relations, recursive query, updatable view, etc.

- Passive data (data are distinct from related functionalities)

- Inefficient support of long-running translations

In order to resolve defects of the Relational data model, some other data models were introduced (*e.g., O.O.* [10] *O. R.* [11], *XML* [12] and *NoSQL* data models [13, 14, 17]).

O.O. (Object-oriented) data model in which representation of data was in form of *classes* and concepts in object oriented paradigm (such as the abstraction, encapsulation and inheritance) was also applied, could not be accepted by developers because of reasons such as its complexity. So, advantages of these two models, the Relational and the object-oriented data models were merged and a new compound, attractive and suitable data model called *Object Relational* (O.R.) was provided. We can say that the traditional Relational model is now extended and nearly substituted by the O.R. data model.

The *O.R.* data model, as a hybrid one, also uses table for its data representation. So, both in both in relational and object-relational models, structure and scheme of data should be predefined and fixed (*i.e., structured* data model). To relax this limitation, the *XML* (as a data model) was proposed. Data representation in XML data model is in form of XML *elements*. Although, semantics of data should be described by an XML element (which is composed of a couple of tags describing their bounding data item), but existence and location of the elements in the XML document is arbitrary (*i.e., semi-structured*). Elements in an XML document implicitly have a tree structure. DBMSs have two options for supporting XML data model; they are either *Native XML* or *XML-enabled*. Both of the Relational and Object-Relational data models, rather than being structured, have another critical defect; they are not *scalable*. In other words, whenever size of the system scales, the performability will be damaged. For example, in a healthcare application, assume that patients' data are stored in a table in O.R. data model. To discover that from whom a specific patient has been infected, we should self-join the patient table for many times in order to compare patients' data and find the first cause of the diseases.

DBMSs, mostly fail to perform such a heavyweight operation, especially when the number of required join operation is too much. So, Relational and Object-Relational data models are not scalable, besides they are week in supporting *relationships*.

NoSQL data models which have better scalability, include *column-based*, *document-based*, *key-value*, and *Graph* data models. The most important advantages of *NoSQL* data

models include (*horizontal*) scalability, to be low-cost, scheme-free, and efficient support of relationships.

Among *NoSQL* models, *Graph* data model is the one the most widely used. The data model proposed in this paper, is also based on the graph model.

In general, *NoSQL* graph data model is based on the graph theory in which, each graph consists of *nodes*, *edges* and their *properties*. A graph database would provide index-free adjacency; each element indicates its related and adjacent element by a direct pointer, without using index and lookup operation. Graph model is very appropriate and efficient especially for applications that have a network essence (*e.g.*, social networking).

As an especial case, in some of applications, the *RDF* (Resource Description Format) is used that is a framework for description of arbitrary resources. Its main application is data fusion in semantic web. RDF [15] is a recommendation of the W3C designed originally to represent metadata. The broad goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines (a priori) the semantics of any application domain.

One of the main advantages (features) of the RDF model is its ability to interconnect resources in an extensible way. Thus, RDF models information with graph-like structure, where basic notions of graph theory like *node*, *edge*, *path*, *neighborhood*, *connectivity*, *distance*, *degree*, *etc.*, play a central role [16].

Existence of the required tools such as *RDF schema*, *OWL* language (which provide ontology definition and complex inductions), *SPARQL* (as a high level query language for graph data) has made RDF usage so pervasive.

Also, there are many data management systems which are briefly introduced in section 5, but none of these data models support big data properly [14]. So, unfortunately, there is no integrated and proper data model for the big data problem and it is an open issue in the context. In the next section, properties of a proper data model for the Big Data problem is discussed and then, the proposed data model is presented.

4. THE PROPOSED DATA MODEL

As stated before, designing proper data model is a fundamental issue in data management and in fact, it is the footstone of the other research activities (*e.g.*, development of system, tools, languages, methods and algorithm). Essentially a data model includes ***(a)*** *data representation,* ***(b)*** *integrity constraints* that should be enforced and satisfied, and ***(c)*** a set of *operators* for data manipulation [9]. As a well-known example, in the Relational data model, data representation is in form of *table* (*i.e.*, a relation which is infect a set of tuples); there are integrity constraints that should be satisfied implicitly (*e.g.*, rows in table as tuples in the relation must be unique, so we need the *key* in each table); and many operators for manipulation of data in tables such as the *Selection*, *Projection*, *Join* and *etc*.. The Relational data model, is appropriate for traditional business applications but not for modern applications such as social networking, since it is structured, is not scalable, and doesn't support relationship, efficiently. Semi-structured data models such as the *XML* and *NoSQL* data models although resolve such defects partially, but they don't satisfy criteria of a proper data model for the big data problem. In order to design a proper data model, first let's discuss important properties of such a data model and then introduce the proposed data model.

4-1- Properties of a proper data model

The most important properties for a proper data model for the big data problem are:

- ***Integrated***

  Although, there exist data models that can be employed separately for each of three dimensions of the big data problem, but we need an integrated data model that can support all of them together, entirely.

- ***Complete***

  It should support all of the required functionality, with no additional thing (*e.g.*, model, structure, language, API, environment, system, tools, etc.) required. Ideally, the model should be *computationally complete* (such as the Object-Relational data model).

- ***Scalable***

  One of the most important defects of the Relational data model is that it is NOT scalable. Scalability for data model means that, whenever size of the problem scales out

for example ***K*** times, system can continue its operation with a tolerable overhead and cost. This is important for the big data problem in which volume of data has a high growth-rate.

- ***Flexible***

  It is also important for data model of the big data problem to be flexible enough to represent various type of data (*e.g., structured, semi-structured, unstructured, etc.*).

- ***Compatible***

  Compatibility with legacy systems and data models makes data integration possible, and also helps interoperability and usability of legacy systems and applications.

- ***Efficient***

  Since data storage (retrieval) in (from) the storage media is really costly and is usually the bottleneck of systems, efficiency is very important in all levels and aspects of data management (especially in data model, as its footstone). For the big data problem in which data in/out must be fast, efficiency is much more critical. In general, efficiency includes both, performance (*e.g.*, response time) and utilization (*e.g.*, memory space). So, a proper data model for the big data should have a good performance and resource utilization.

4-2- Design principles

In order to design a data model for big data that can properly satisfy the properties mentioned above, some principle must be conformed. First of all, the designed data model must adhere and *be compatible* with the previous data models (*e.g.*, use them as the design primitives). Accordingly, as is stated later, the proposed data model uses *O.R.*, *XML* and *NoSQL* data models. Selection of the proper base-model and required properties is described briefly below.

Among the *NoSQL* data models (that generally are scalable and flexible), the *Graph* data model is used more in new applications [19, 20]. It is also more powerful than its similar data models (for example *tree-like* models such as the XML data model, since it provides navigation in all directions (rather than parent-to-child direction)). So, the graph data model [19, 20] is selected as the base data model to be modified such that the required properties can be satisfied in the proposed data model.

The other design principle is *abstraction* (in the proper level). Since the variety is one of the important characteristics of big data, in order to be flexible enough to handle various types of data (*e.g.*, structured, semi-structured and unstructured data) together in one integrated data model, proper abstraction level must be determined and conformed.

Finally, an integrated proposed data model obviously must be *correct* and *ambiguity-free*. So, required implicit integrity constraints for the new data model must be determined and enforced.

Additionally, it is better that the proposed data model to be *computationally complete*.

Accordingly, the proposed data model which is based on the *NOSQL Graph* data model and modified to satisfy the mentioned required properties is introduced below.

4-3- The proposed *Hyper Nested Graph* (*HNG*) data model

As discussed before, a data model includes *data representation*, set of implicit *integrity constraints*, and set of *operators*.

(a) Data representation

The proposed data model is based on the graph data model [19, 20]. A graph $G =< V, E, D, L, W >$ includes *V*: set of nodes, *E*: set of edges, *D*: set of direction of edges, *L*: labels of nodes or edges, and *W*: weights of nodes or edges. Extended versions of basic graph include *nested graph* (*i.e.*, contains hyper nodes), *hyper graph* (contains hyper edges), and *attributed graph*. The proposed data model is an extended graph model in which both nodes and edges are hyper and nested (so, called ***Hyper Nested Graph (HNG)***). By nested node, we mean that each node can contain one or some nodes inside; and by nested edge we mean that an edge is an abstract form some links between its corresponding nodes, each one indicating one relationship between its source and destination nodes (figure 3).

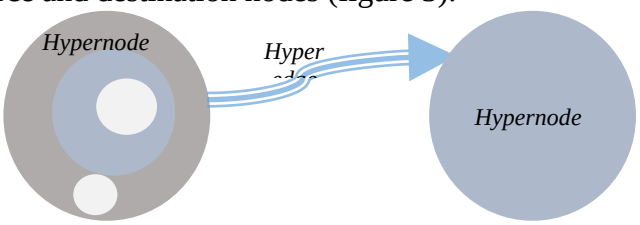
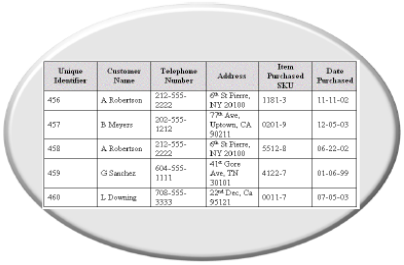


Fig. 3: schematic view of *HNG*

For both nodes and edges in *HNG*, a header part contains metadata while the context part, contains

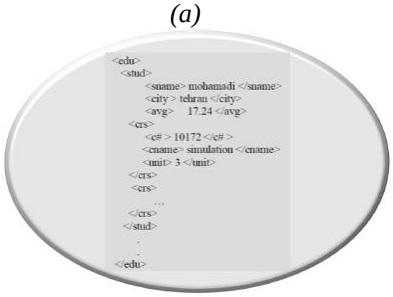
the content of the object (*i.e.*, node or edge). *O-id* (*Object-identifier*) *O-name* (*Object-name*) are some of the most important fields in the header part of each object. Object identifier (which must be unique) can be generated and handled in of these ways: *user generated*, *system generated*, or *derived* from object's primary key (whenever the object is a node containing data tuples) or URI (when the object is a resource in web, *e.g.*, a web page or multimedia file).

Also, when the object is an edge, header part can contain metadata such as the unidirectional/bidirectional, weight, priority, control data, etc. The header part of objects can be semi-structured, as an XML element. The content part for each node (as a nested node) contains content in forms of structured data (*e.g.*, value of an attribute for an entity instance, a tuple of attributes (*i.e.*, a row), a table, a nested table, etc.), semi-structured data (*e.g.*, an XML element which itself can contain nested elements), unstructured data (*e.g.*, documents in various types including text, voice, image, video, etc.), or event can contain one or more other nodes inside it (*recursive definition*) (figure 4).



Unique Identifier	Customer Name	Telephone Number	Address	Item Purchased SKU	Date Purchased
456	A Robertson	212-555-2222	6th St Fierre, NY 20160	1181-3	11-11-02
457	B Meyers	209-555-1212	77th Ave, Uptown, CA 90211	0201-9	12-05-03
458	A Robertson	212-555-2222	6th St Fierre, NY 20160	5512-8	06-22-02
459	G Sanchez	604-555-1111	41st Glore Ave, TN 30101	4122-7	01-06-99
460	L Downing	708-555-3333	22nd Dec, Ca 90121	0011-7	07-05-03

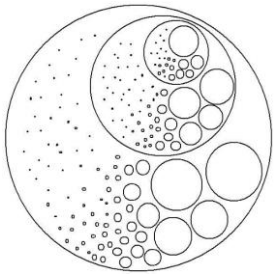
(a)



```
<edu>
  <stud>
      <sname> mohamadi </sname>
      <city > tehran </city>
      <avg>   17.24 </avg>
    <crs>
      <c# > 10172 </c# >
      <cname> simulation </cname>
      <unit> 3 </unit>
    </crs>
    <crs>
      ...
    </crs>
  </stud>
  .
  .
</edu>
```

(b)

(c)



(d)

Fig. 4: example of *(a)* structured data, *(b)* semi-structured data, *(c)* unstructured data, and *(d)* nested node

In order to establish a connection between a node to its nested nodes, *ref* (reference) data type can be used which is in fact a pointer to the corresponding object. This nested and hierarchical structure is so common and well-known that is employed in many different contexts (*e.g.*, file system in operating systems to manage directories, sub-directories, and files). Content part of a nested edge which represent relationships between the nodes also can be implemented as an XML element.

(b) Integrity constraints

In general, integrity constraints are classified into *explicit* constraints (which are explicitly defined by developers, and are application-dependent), and *implicit* constraints (which must be enforced implicitly by the system on *every* database which use that data model). For example, in the conventional Relational data model, the need for a primary key for each of tables is an implicit integrity constraint while for instance, $18 \leq age \leq 36$ for table of *students* is an explicit constraints which many not be held for *age* attribute of other tables, *e.g.*, *employee*.

On the other hand, integrity constraints can be grouped in *schema-instance* consistency, *identity* and *referential* integrity, and *functional* and *inclusion dependencies*. Examples of these are, labels with unique names [21], typing constraints on nodes [22], functional dependencies [23], domain and range of properties [24].

Designing a data model, its implicit integrity constraints also must be determined and defined by the designers and enforced by the systems. The most essential implicit integrity constraints that must be held and enforced for every database that uses the proposed hyper nested graph data model, can be classified into these two classes: those that are *generic* for all graph-based data models, and

those that are *specific* to the proposed hyper nested graph.

Some of generic graph-based data models' integrity constraints are [16]:

- *Schema instance consistency*
- *Identification of models, attributes, and relations*
- *Path constraints*

which must be enforced in *HNG* data model, since it is also graph-based.

Moreover, due to the modifications and extensions applied for preparing the proposed *HNG* data model, these *HNG-specific* integrity constraints also must be enforced implicitly.

i) Finite nesting

Because of the recursive nature of *HNG's* definition (in which each node itself can contain node(s)), similar to other recursive equations, we need boundary conditions: *nesting* (of nodes) *must be finite* (*i.e., non-endless*).

Although each node can contain contents (*e.g.*, structured, semi-structured or unstructured data), or can contain other nodes inside, but the number of this nesting must be bounded. This is because systems that are used in practice, be able to implement and handle such nested nodes. As an example, suppose that the maximum number of nesting allowed for a node is set to 32, as one of the data management products limitations.

Figure 5 illustrates a schematic example of an infinite nested hypernode.

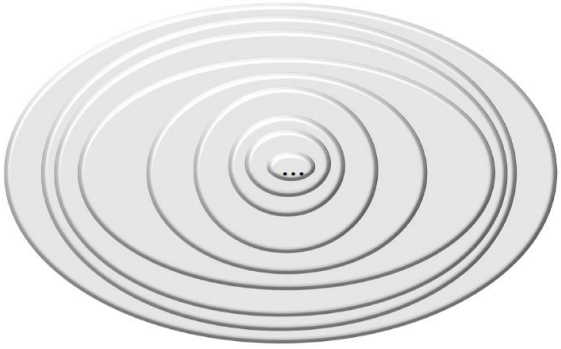


Fig. 5: schematic example of an infinite nested node

ii) Acyclic nesting

In general, *cycle* (loop) in *HNG* is not forbidden as an implicit integrity constraint and *inter-nodes cycle* is allowed (it may be used for modeling and representing relationships between objects in the real word). But, nesting of the nodes in *HNG* must be *acyclic*. In other words, references to the nodes inside a particular node named ***N*** must not be such that return to the ***N*** itself. Such *intra-node cycle* (figure 6) which constructs an infinite loop is not meaningful and is not event acceptable in theory. So, intra-node cycle is not allowed.

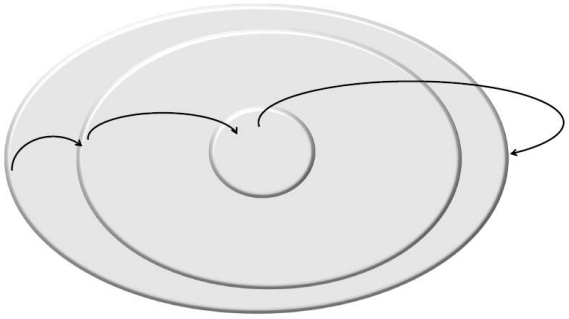


Fig. 6: schematic example of an intra-node cycle

iii) Referential integrity

Similar to the referential integrity in the Relational data model, references in *HNG* data model must be *valid*. So, dangling, missed, or wrong references must be prevented. Options such as *restricted*, *cascade*, *trigger* or even *no-action* can be implemented and used in case of deletion or update of references. As an example, restricted option would *prevent* the devastative modification (delete/update) on the referred node, while the cascade option would *substitute* the updated/deleted node by its (logical/structural) successor or predecessor.

As discussed above, implicit integrity constraints of a data model must be implemented and enforced by the underlying systems (*e.g.*, DBMS) for every database which use that data model. So, data management systems that would use the proposed *HNG* data model must implement and enforce such integrity constraints.

Providing efficient practical solutions for enforcing such integrity constraints can be done as future work.

5. EXPERIMENTAL EVALUATION

Evaluation of the proposed *Hyper Nested Graph* (*HNG*) data model for big data problem is performed experimentally via implementation and practical comparison of *HNG* with the other alternatives. Since the proposed *HNG* data model is a graph-based model, *Neo4j* which is an open-source graph-based data management system is used for extension and implementation of *HNG* data model.

5-1- Experimental setup

YCSB (Yahoo! Cloud Serving Benchmark) [26, 27], which is used as data set and query set for

this experimental evaluation is a standard benchmark publicly available. Different CRUD operations (such as insert, update, and query), and also a mixture of such operations are executed on a data set of the big data and some desired metrics are measured. The underlying machine has platform of Linux Ubuntu 12.04 and SuperMicro server with 8 cores and 40 GB RAM. The proposed Hyper Nested Graph (*HNG*) data model is compared with the *document-based* (*e.g., MongoDB* [28]), *Graph document-based* (*e.g., OrientDB* [29]), and *Graph* (*e.g., Neo4j* [30]) *NoSQL* data models.

Average case for several runs of different scenarios is computed for the most important parameters such as *response-time* in the *best-case* and *average-case*. Also, two other metrics that are defined in YCSB are measured and illustrated. The metric is defined as *Response time*: The time interval from submission of a request to its completion.

5-2- Experimental results

Using the YCSB [26, 27] as the tested, data sets with 10e3, 10e6 and 10e10 data items were generated, each one containing 1*KB* data with 10 different fields and different CRUD operations run on them [17]. Average values for desired metrics are computed and illustrated in figures below to compare different data models. Figures 5 through 7 show response time of compared data models for different operations (each using a data set with 10e3, 10e6, and 10e10 data items, respectively).

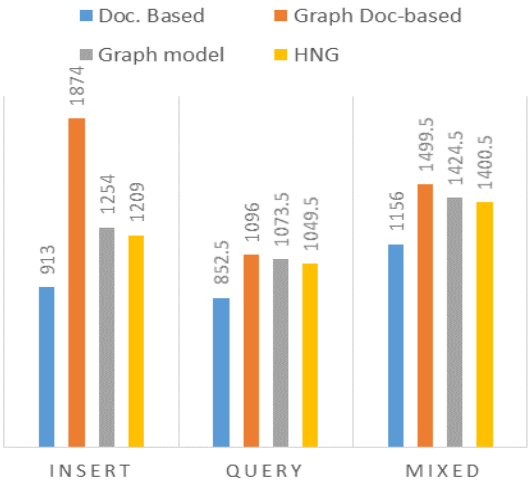


Fig. 7: response time of different operations for 10e3 data items

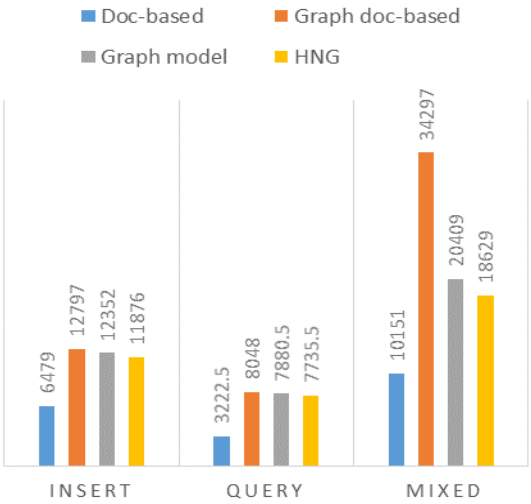


Fig. 8: response time of different operations for 10e6 data items

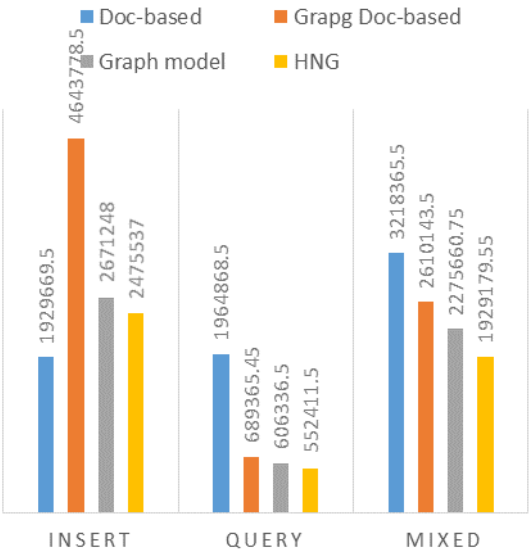


Fig. 9: response time of different operations for 10e10 data items

Summarizing, as is shown if figures 7 through 9, although the document-based data model has better efficiency when handling small data sets, graph-based data models outperform document-based model in terms of response time while the size of data sets increases.

Also, the proposed *Hyper Nested Graph* (*HNG*) data model has better performance rather than the pure graph data model and graph document-based data model. It can be induced from these practical experiments that the proposed *HNG* data model will have an acceptable efficiency when used for managing Big Data.

6. CONCLUSION AND FUTURE WORK

A proper designed data model should have some characteristics. The most necessary ones for a

data model designed for the Big Data problem are to be *integrated*, *complete*, *scalable*, *flexible*, *compatible*, and *efficient*.

In this paper, a proper data model for big data is designed and presented. In the proposed ***HNG*** (*Hyper-Nested Graph*) data model that is based on the *NoSQL Graph* data model and extended to meet the required conditions and characteristics, both nodes and edges are nested. Sub-nodes of each node that can be accessed via a *ref* (reference type) can store either *structured* (*e.g.*, *scalar-value*, *row*, or *table*), *semi-structured* (*e.g.*, an *XML element* or *document*), or *unstructured* (*e.g.*, *text*, *image*, *voice*, or *video*) data.

Choosing such level of abstraction provides *compatibility* with the older data models, beside a good *flexibility*.

Powerful support of relationships in Graph-based data models, beside fast and convenient Travers capability make it possible to have a good *efficiency* and *scalability* while removing the defect of traditional data models (e.g., the *Relational*) in supporting relationships.

Nesting, which is used in *HNG* (to overcome being big in the Big Data problem), is also a well-known approach in many other contexts (*e.g.*, file system of OSs) and there exist many good practices even for its implementation.

Experimental evaluation results show that the proposed *HNG* data model outperforms other data models which are usually used (e.g., document-based, Graph Document-based, and Graph data models), in terms of response time.

Data model is the footstone of data management activities. So, for each new data model, such activities can be followed. Some of such future works include:

- Implicit integrity constraints for the *HNG*
- Set of required operators
- Providing theoretical and formal tools such as *HNG algebra* (similar to the *Relational Algebra*)
- SQL-like database language for *HNG*
- Developing data management systems and tools supporting *HNG* as data model

Moreover, issues such as *optimization*, *security*, *transaction management* (*ACID*-compliant or *CAP*-based) can be followed for completing *HNG* data model and employing in real-world Big Data applications.

References

[1] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, Angela Hung Byers, Big data: The Next Frontier forInnovation, Competition, and Productivity, McKinsey Global Institute, 2012 .

[2] Gantz J, Reinsel D (2011) Extracting value from chaos. IDC iView, pp 1–12 .

[3] Philip Chen, C. L., and Chun-Yang Zhang. "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data." Information Sciences 275 (2014): 314-347.

[4] Chen, Min, Shiwen Mao, and Yunhao Liu. "Big Data: A Survey." Mobile Networks and Applications 19.2 (2014): 171-209.

[5] Codd, E. F. 1970. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 13, 6, 377–387.

[6] Codd, E. F. 1983. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 26, 1, 64–69.

[7] Tsichritzis, D. C. and Lochovsky, F. H. 1976. Hierarchical Data-Base Management: A Survey. ACM Computing Surveys 8, 1, 105–123.

[8] Taylor, R. W. and Frank, R. L. 1976. CODASYL Data-Base Management Systems. ACM Computing Surveys 8, 1, 67–103.

[9] Codd, E. F. 1980. Data Models in Database Management. In Proc. of the 1980 Workshop on Data abstraction, Databases and Conceptual Modeling. ACM Press, 112–114.

[10] Kim, W. 1990. Object-Oriented Databases: Definition and Research Directions. IEEE Transactions on Knowledge and Data Engineering (TKDE) 2, 3, 327–341.

[11] Nori, Anil, et al. "Data model for object-relational data." U.S. Patent Application 11/228,731.

[12] Buneman, P. 1997. Semistructured Data. In Proc. of the 16th Symposium on Principles of Database Systems (PODS). ACM Press, 117–121.

[13] Bray, T., Paoli, J., and Sperberg-McQueen, C. M. Extensible Markup Language (XML) 1.0, W3C Recommendation 10 February 1998. http://www.w3.org/TR/1998/REC-xml-19980210.R.

[14] Han, Jing, et al. "Survey on NoSQL database." *Pervasive computing and applications (ICPCA), 2011 6th international conference on*. IEEE, 2011.

[15] Klyne, G. and Carroll, J. 2004. Resource Description Framework (RDF) Concepts and Abstract Syntax.

http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[16] Angles, Renzo, and Claudio Gutierrez. "Survey of graph database models." *ACM Computing Surveys (CSUR)* 40.1 (2008): 1.

[17] Hecht, Robin, and S. Jablonski. "NoSQL Evaluation." *International Conference on Cloud and Service Computing*. 2011.

[18] Ghrab, Amine, et al. *Analytics-Aware Graph Database Modeling*. Technical report, 2014.

[19] Kunii, Hideko S. "Graph Data Model." *Graph Data Model*. Springer Japan, 1990. 7-20.

[20] Robinson, Ian, Jim Webber, and Emil Eifrem. *Graph databases*. " O'Reilly Media, Inc.", 2013.

[21] Vicknair, Chad, et al. "A comparison of a graph database and a relational database: a data provenance perspective." *Proceedings of the 48th annual Southeast regional conference*. ACM, 2010.

[22] Kuper, G. M. and Vardi, M. Y. 1993. The Logical Data Model. ACM Transactions on Database Systems (TODS) 18, 3, 379–413.

[23] Levene, M. and Poulovassilis, A. 1991. An Object-Oriented Data Model Formalised Through Hypergraphs. Data & Knowledge Engineering (DKE) 6, 3, 205–224.

[24] Gyssens, M., Paredaens, J., den Bussche, J. V., and Gucht, D. V. 1990. A Graph-Oriented Object Database Model. In Proc. of the 9th Symposium on Principles of Database Systems (PODS). ACM Press, 417–424.

[25] Buerli, Mike, and C. P. S. L. Obispo. "The current state of graph databases." *Department of Computer Science, Cal Poly San Luis Obispo, mbuerli@ calpoly. edu* (2012): 1-7.

[26] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears: *Benchmarking Cloud Serving Systems with YCSB*,Yahoo! Research,Santa Clara, CA, USA

[27] Cooper, Brian F., et al. "*Benchmarking cloud serving systems with YCSB*."*Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010.

[28] Chodorow, Kristina. *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

[29] Developers, OrientDB. "OrientDB." Hybrid Document-Store and Graph NoSQL Database {online} (2012).

[30] Miller, Justin J. "Graph Database Applications and Concepts with Neo4j." Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA March 23rd-24th. 2013.

[31] Allen, Sharon, and Evan Terry. *Beginning relational data modeling*. Apress, 2005.

[32] Navathe, S. B. 1992. Evolution of Data Modeling for Databases. Communications of the ACM 35, 9, 112–123.

[33] Hull, R. and King, R. 1987. Semantic Database Modeling: Survey, Applications, and Research Issues. ACM Computing Surveys 19, 3, 201–260.

[34] Ter Bekke, Johan H., and J. H. Ter Bekke. *Semantic data modeling*. Hemel Hempstead: Prentice Hall, 1992.

[35] Hull, R. and King, R. 1987. Semantic Database Modeling: Survey, Applications, and ResearchIssues. ACM Computing Surveys 19, 3, 201–260.

[36] Poulovassilis, A. and Levene, M. 1994. A Nested-Graph Model for the Representation and Manipulation of Complex Objects. ACM Transactions on Information Systems (TOIS) 12, 1, 35–68.

[37] Levene, M. and Loizou, G. 1995. A Graph-Based Data Model and its Ramifications. IEEE Transactions on Knowledge and Data Engineering (TKDE) 7, 5, 809–823.

[38] Levene, M. and Poulovassilis, A. 1990. The Hypernode Model and its Associated Query Language. In Proc. of the 5th Jerusalem Conf. on Information technology. IEEE Computer Society Press, 520–530.

[39] Tuv, E., Poulovassilis, A., and Levene, M. 1992. A Storage Manager for the Hypernode Model. In Proc. of the 10th British National Conference on Databases. Number 618 in LNCS. Springer-Verlag, 59–77.

[40] Berge, C. 1973. Graphs and Hypergraphs. North-Holland, Amsterdam.

[41] Wood, Peter T. "Query languages for graph hdatabases." *ACM SIGMOD Record* 41.1 (2012): 50-60.

[42] McColl, Robert Campbell, et al. "A performance evaluation of open source graph databases." *Proceedings of the first workshop on Parallel programming for analytics applications*. ACM, 2014.

[43] Andries, M., Gemis, M., Paredaens, J., Thyssens, I., and den Bussche, J. V. 1992. Concepts for Graph-Oriented Object Manipulation. In Proc. of the 3rd Int. Conf. on Extending Database Technology (EDBT). LNCS, vol. 580. Springer, 21–38.

[44] Mainguenaud, M. 1992. Simatic XT: A Data Model to Deal with Multi-scaled Networks. Computer, Environment and Urban Systems 16, 281–288.

[45] Amann, B. and Scholl, M. 1992. Gram: A Graph Data Model and Query Language. In European

Conference on Hypertext Technology (ECHT). ACM, 201–211.

[46] Gemis, M. and Paredaens, J. 1993. An Object-Oriented Pattern Matching Language. In Proc. of the First JSSST Int. Symposium on Object Technologies for Advanced Software. Springer-Verlag, 339–355.

[47] Hidders, J. and Paredaens, J. 1993. GOAL, A Graph-Based Object and Association Language. Advances in Database Systems: Implementations and Applications, CISM, 247–265.

[48] Graves, M., Bergeman, E. R., and Lawrence, C. B. 1995a. A Graph-Theoretic Data Model for Genome Mapping Databases. In Proc. of the 28th Hawaii Int. Conf. on System Sciences (HICSS). IEEE Computer Society, 32.

[49] Hidders, J. 2002. Typing Graph-Manipulation Operations. In Proc. of the 9th Int. Conf. on Database Theory (ICDT). Springer-Verlag, 394–409.

[50]Safaei Ali A. 2016. Real-time Processing of Streaming Big Data. Journal of Real-time Systems (Real-Time Syst.) 52, 2, 1-44. DOI 10.1007/s11241-016-9257-0.