ISSN 0146-4116, Automatic Control and Computer Sciences, 2016, Vol. 50, No. 7, pp. 493–496. © Allerton Press, Inc., 2016.
Original Russian Text © A.G. Marchuk, 2014, published in Modelirovanie i Analiz Informatsionnykh Sistem, 2014, Vol. 21, No. 6, pp. 169–175.

PolarDB: An Infrastructure for Specialized NoSQL Datebases and DBMS

A. G. Marchuk

*Ershov Institute of Informatics Systems, Siberian Branch, Russian Academy of Sciences,
pr. Lavrent'eva 6, Novosibirsk, 630090 Russia*

e-mail: mag@iis.nsk.su

Received October 3, 2014

Abstract—This paper presents a new infrastructure for creating specialized databases and database management systems (DBMSs). Some principles of this infrastructure are formulated based on an analysis of various NoSQL solutions. The main features of the proposed approach are: (a) a flexible system of type definitions that allows one to create data structures based on different paradigms and (b) different forms of supporting structured data and mapping them onto the file system. Experiments are carried out to implement RDF graphs, relational tables, name tables, and object-relational mappings. The proposed approach allows one to solve certain problems of developing new technologies for working with Big Data.

Keywords: PolarDB, Big Data, NoSQL, RDF, databases, DBMS

DOI: 10.3103/S0146411616070142

INTRODUCTION

While creating electronic archives and solving various problems of historical factography [1], the Institute of Informatics Systems of the Russian Academy of Sciences has accumulated valuable experience in processing RDF data. RDF is quite a universal method for data structuring that is oriented to creating databases of various types. In addition, the RDF representation corresponds to a directed graph, which makes it possible to use RDF for solving various problems of discrete mathematics. In late 2012 and early 2013, experiments were carried out in order to compare the capabilities of universal and specialized database management systems (DBMSs) in implementing RDF.

The DBMSs used in the experiments [2], including relational (MS SQL Server and MySQL) and NoSQL (MongoDB and Cassandra) [3] DBMSs, were found to be not quite suited to this task [2]. All of these solutions were marked bad or very bad in processing large volumes of data (>100 million arcs per graph). Only Open Link Virtuoso [4], which is oriented to RDF, showed more or less acceptable results; however, it is a commercial DBMS and its freeware version will not save developers from unpleasant surprises in the future. Based on these findings, an experiment was carried out in order to construct an RDF data storage on a more primitive basis (binary files). As a result, a new specialized DBMS was created and tested on a real dataset containing one billion triples (arcs). This solution showed promising results and, in mid-2013, a project was initiated for creating the PolarDB infrastructure oriented to designing specialized DBMSs and Big Data processing.

1. PRINCIPLES OF POLARDB

Below are the basic principles underlying the PolarDB infrastructure.

1. The database is implemented using instruments of the file system.

2. All data should be typified and flexible data structuring should be provided.

3. The performance is improved by reducing the number of read/write operations and by minimizing the movement of the pointer "Position."

4. Stream processing should be used to the fullest extent possible.

The data structuring system is based on the type system of the Polar language [5]. It includes the following basic elements.

A structured value is a treelike structure (value) interpreted in terms of a given type.

A type is a treelike structure that sets the interpretation of structured values (the type can be represented as a structured value).

The structured values are mapped onto the storage system using a system tools of the file system. In turn, this mapping is optimized using the system tools of the operating system. This allows one to take advantage of the mechanisms used in modern operating systems and modern architectures to accelerate the processing of disks and disk files while avoiding the interference of different cache structures.

2. TYPE SYSTEM

The type can be primitive, atomic, constructible, or string.

Primitive types are boolean, character, integer, longinteger, real, and none (the set of values that contains no elements). A string type is sstring (corresponds to the string in object-oriented programming).

Constructible types are

• record: a fixed set of typified fields; fields 0, 1, ..., n – 1 represent the value of the corresponding type specified in the definition of the record (the number and types of fields are fixed);

• sequence: an ordered set containing an indeterminate (zero or more) number of like elements;

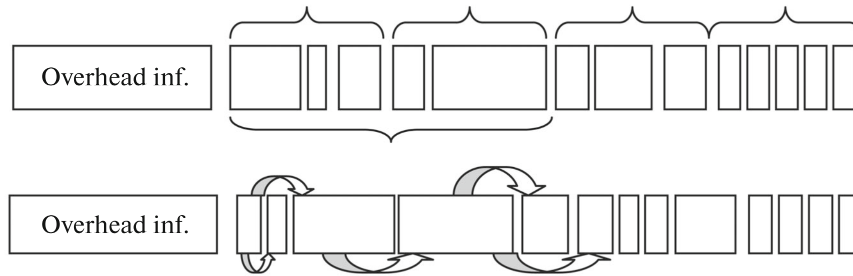
• union: a value that consists of a tag and a subvalue; the tag (dynamically) sets the type of the subvalue (tags are numbered beginning with zero).

The type system is given by the abstract class PType. Below is an example of defining a system of types that correspond to RDF representations.

```
PType tp_rec = new PTypeRecord(
 new NamedType("f1", new PType(PTypeEnumeration.integer)),
  new NamedType("f2", new PType(PTypeEnumeration.sstring)),
  new NamedType("f3", new PType(PTypeEnumeration.longinteger)));
PType tp_seq = new PTypeSequence(new PTypeRecord(
 new NamedType("id", new PType(PTypeEnumeration.sstring)),
  new NamedType("name", new PType(PTypeEnumeration.sstring)),
   new NamedType("fd", new PType(PTypeEnumeration.sstring)),
    new NamedType("deleted", new PType(PTypeEnumeration.boolean))));
Ptype  seqtriples = new PTypeSequence(
  new PTypeUnion(
    new NamedType("empty", new PType(PTypeEnumeration.none)),
      new NamedType("op",
        new PTypeRecord(
          new NamedType("subject", new PType(PTypeEnumeration.sstring)),
new NamedType("predicate", new PType(PTypeEnumeration.sstring)),
          new NamedType("obj", new PType(PTypeEnumeration.sstring)))),
          new NamedType("dp",
            new PTypeRecord(
new NamedType("subject", new PType(PTypeEnumeration.sstring)),
new NamedType("predicate", new PType(PTypeEnumeration.sstring)),
            new NamedType("data", new PType(PTypeEnumeration.sstring)),
new NamedType("lang", new PType(PTypeEnumeration.sstring))))));
```

3. STRUCTURED VALUES

The structured values of the given types are stored (packed) in cells. The cells act as variables, so storing a structured value in a cell is similar to storing values in variables of traditional programming languages. The difference is that, upon executing a program, the value continues to exist. The cells are implemented as byte streams, generally in the form of direct-access binary files.



Structure of the cells of free and fixed formats.

The figure shows the cells of free and fixed formats. In both the cases, the cell is a byte stream preceded by some overhead information. The difference is in the location of structured values. For free-form cells, the hierarchy of structured values unfolds into a serialized flow of subvalues. For fixed-form cells, references between information blocks are used.

Structured values can be represented as text. The textual representation of the values of primitive types is traditional. The record is represented as {field1, field2, ...}, the sequence as [element1, element2, ...], and the union as tag_name^value.

For example, the value of the type seqtriples can be represented as

```
[
  op^{"fogid.net/e/id3kjf", "rdf:type", "fogid.net/o/person"},
  dp^{"fogid.net/e/id3kjf", "fogid.net/o/name", "Иванов И.И.", "ru"},
  op^{"fogid.net/e/id3kjf", "fogid.net/o/father", "fogid.net/e/d9fdjf"}
]
```

Structured values can also be represented as objects in the random-access memory. In this case, the primitive types are represented as system types (bool, int, long, and string), the record can be represented as an array of objects (object[]), the sequence can be represented as an array of objects, and the union can be represented as an array of objects (an array of two elements: tag and value of a variant).

Using Linq, the object representation can easily be generated (e.g., from XML) and transformed into other representations. The object form can be written—Set()—to the input of the cell and can be read—Get()—from its output.

The elements of the structured values stored in cells are accessed via entries. The corresponding construction is given by the class PaEntry for free-form cells and by the class PxEntry for fixed-form cells. The root (whole) value is accessed via the field “Root.” In the fragment

```
PxCell xcell = new PxCell(type, path, true);
PxEntry ent2 = xcell.Root;
```

the first operator defines the cell and its connection with the file, while the second operator defines the entry corresponding to the root value stored in the cell.

```
The class PxEntry includes the following methods:
PxEntry Field(int nfield);
Int64 Count();
PxEntry Element(long ind);
IEnumerable<PxEntry>Elements() // Creates a flow of record elements
Int Tag();
PxEntry UElement();
object Get();
void Set(object value).
```

These methods set the entries that correspond to subvalues of structured values and allow one to read and write these values. A similar set of methods is defined for the class PaEntry.

One of the problems the DBMS developers often does not take into account the input of data, especially Big Data. For relational DBMSs, it is believed to be sufficient to add an entry into a predefined table.

In our case, the structure of the stored data can differ; therefore, special methods have been developed for entering the data into the cells. For free-form cells, the basic method is the so-called bracketed serial flow (of events), which can be illustrated based on the example of specifying the interface of such a flow as follows:

```
public interface ISerialFlow
{
  void StartSerialFlow();
  void EndSerialFlow();
  PType Type  get;
  void V(object val);
  void R(); void Re();
  void S(); void Se();
  void U(int tag);
  void Ue();
}
```

This group of methods is defined for free-form cells; for fixed-form cells, the polish structural flow is used. A polish structural flow is a flow of primitive elements, object representations of structures, unfolded sequences preceded by a number of elements, the sign of record unfolding, and unfolded records; the union is represented as a pair of a tag and a subvalue.

In addition, various sorting and binary search methods have been implemented for the efficient processing of Big Data.

CONCLUSIONS

Presently, the PolarDB infrastructure is implemented and used to construct experimental DBMSs for various applications. The following experiments have been carried out:

(1) processing RDF databases when working with electronic archives and solving problems of historical factography;

(2) checking for the ability of processing relational tables;

(3) testing an experimental RDF-Sparql system aimed at achieving the maximum efficiency of RDF processing;

(4) checking for the ability of implementing equilibrium binary trees;

(5) implementing (large) name tables;

(6) constructing ORM-like object-relational representations.

REFERENCES

1. Marchuk, A.G. and Marchuk, P.A., A platform for implementation of electronic archives of data and documents, *Elektronnye biblioteki: Perspektivnye metody i tekhnologii, elektronnye kollektsii: Trudy XIV Vserossiyskoi nauchnoi konferentsii RCDL'2012* (Digital Libraries: Advanced Methods and Technologies, Digital Collections: Proc. XIV All-Russian Sci. Conf. RCDL'2012), Pereslavl-Zalessky, 2012, pp. 332–338.
2. Marchuk, A.G., On the way to large RDF data, *Elektronnye biblioteki: Perspektivnye metody i tekhnologii, elektronnye kollektsii: Trudy XIV Vserossiyskoi nauchnoi konferentsii RCDL'2013* (Digital Libraries: Advanced Methods and Technologies, Digital Collections: Proc. XV All-Russian Sci. Conf. RCDL'2013), Yaroslavl, 2013, pp. 51–56.
3. NoSQL: Ultimate Guide to the Non-Relational Universe. http://nosql-database.org/.
4. Open Link Software. http://www.openlinksw.com/.
5. Marchuk, A.G. and Lel'chuk, T.I., *Yazyk programmirovaniya Polyar: Opisanie, ispol'zovanie, realizatsiya* (Programming Language Polyar: Description, Use, and Implementation), Novosibirsk, 1986.

Translated by Yu. Kornienko