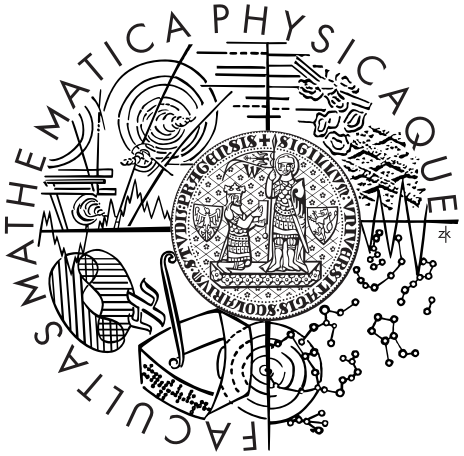
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Marcel Kikta

Evaluating relational queries in pipeline-based environment

Department of Software Engineering

Supervisor of the master thesis: David Bednárek

Study programme: Software systems

Specialization: Software engineering

Prague 2014

Dedication.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Název práce: Vyhodnocování relačních dotazů v proudově orientovaném prostředí

Autor: Marcel Kikta

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. David Bednárek, Ph.D.

Abstrakt:

Klíčová slova: SQL, Překladač, Relační algebra, Optimalizator, Bobox

Title:

Author: Marcel Kikta

Department: Název katedry či ústavu, kde byla práce oficiálně zadána

Supervisor: RNDr. David Bednárek, Ph.D.

Abstract:

Keywords: SQL, Compiler, Relational algebra, optimizer, Bobox

Contents

Introduction

Today's processors have multiple cores and it's single core performance is improving only very slow because of physical limitations. On the other hand number of cores is still increasing and we can assume that it will continue. That's why developing parallel software is crucial for improving overall performance.

Parallelization can be achieved manually or using some framework designed for it. For example there are frameworks like OpenMP or Intel TBB. Department of Software Engineering at Charles University in Prague developed it's own parallelization framework called Bobox[1].

Bobox is designed for parallel processing large amounts of data. It was specifically created to simplify and speed up parallel programming of certain class of problems - data computations based on non-linear pipeline. It was created to evaluate queries over relational data but it was succesfully used in implementation of XQuery and TriQuery engines.

Bobox contains from runtime environment and operators. Theses operators are called boxes and they are C++ implementation of data processing algorithm. Boxes use messages called envelopes to send processed data to each other.

Bobox takes as input execution plan written in special language Bobolang[2]. It allows to define used boxes and simply connect then into directed acyclic graph. Bobolang specifies the structure of whole application and also the inner structure of each box. It can create highly optimized evaluation, which is capable of using the most of the hardware resources. The language has been tested in several applications and it turned out to be very powerful tool in data processing massive parallel application.

Most used databases are relational databases. They are based on the view of data organized in tables called relations. SQL[3] ("Structured query language") is very important language based on relation databases. It is used for queering data, modifying content of tables and also the structure of tables. When we want to evaluate query we need to parse query text input into parse tree. This form will be transformed to relational algebra, which we call logical query plan. It will be optimized and physical plan is generated. Physical plan indicates not only operation performed, but also which order are they performed and what kind of algorithms are used for execution.

The main goal of this thesis is to implement part of SQL compiler. The input is query written in XML format in from of relational algebra. Program validates input, optimizes and transforms it to physical plan of given query. The output is execution plan for Bobox written in Bobolang.

1. Architecture

1.1 Bobox

In the section we describe basic architecture of Bobox. Information source for this chapter is Doctoral thesis Parallel Processing of Data[4].

Overall Bobox architecture is displayed in figure 1.1. Framework contains of Boxes. Box is basically a C++ class containing implementation of data processing algorithm or it can be set of connected boxes. Box can have arbitrary number of inputs and outputs. All boxes are connected to a directed acyclic graph.

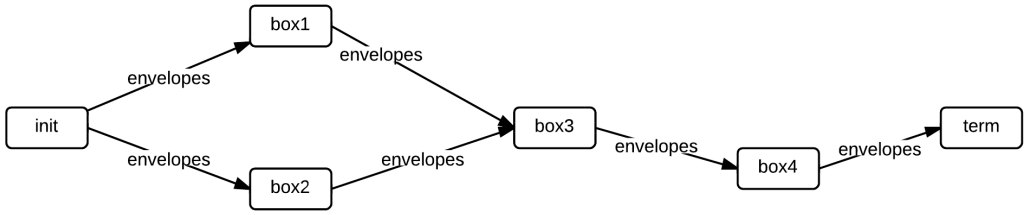


Figure 1.1: Bobox architecture.

Data streams are implemented as data units called enveloped. Envelope structure is displayed in figure 1.2. It consists of sequence tuples, but internally data are stored by columns, that means envelope contains from sequence of columns and it's data is stored in separate list. So to read all attributes of the i-th tuple we have to access all column lists and read it's i-th element. There is special type of envelope having poisoned pill. It is send after all valid data indicating end of data stream.

There are two special boxes, which have to be in every execution plan:

- *init* - first box in topological order and it indicates starting box of execution plan
- *term* - last box in topological order and indicates that plan has been completely evaluated

Evaluation starts with scheduling *init* box, which sends poisoned pills to all of its output. All of it's output boxes will be scheduled. They can read data from hard drive or network, process it and sent it to other boxes for further processing. Other boxes usually receives data in envelopes in their inputs. Box *term* waits for every it's input to receive poisoned pill and then evaluation ends.

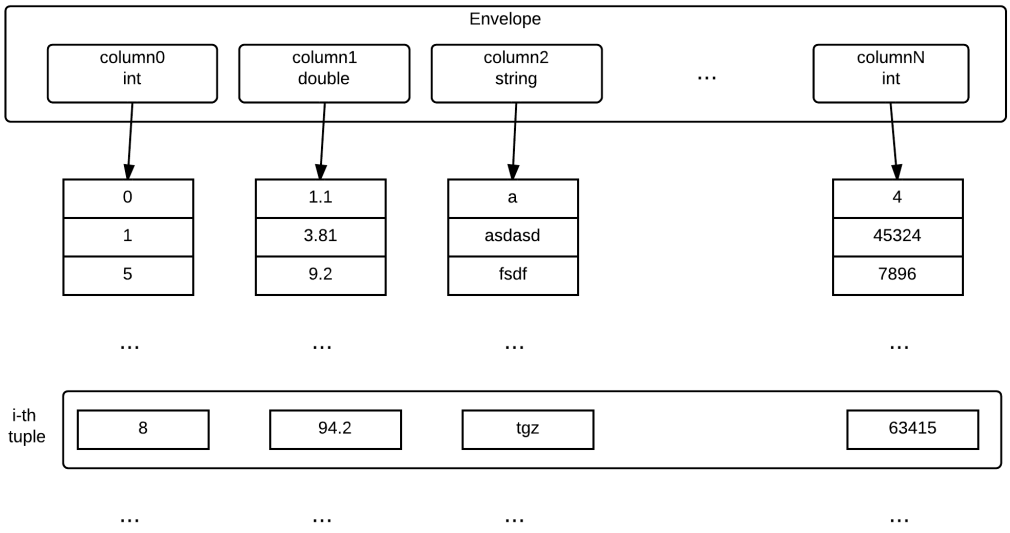


Figure 1.2: Envelope structure.

1.2 Bobolang

In this section we describe syntax and semantics of Bololang language. We used paper Bobolang - a language for parallel streaming applications[2] as information source.

Bobolang is a formal description language for Bobox execution plan. Bobox environment provides implementation of basic operators (boxes). Bobolang let's programmer choose which boxes to used, what boxes to use, what type are passed and how the boxes are interconnected. Bobolang also provides possibility to create operators connecting existing ones.

In following example we show a definition using other operators:

```
operator process (int)->(int,int,int)
{
    preproc(int)->(int,int) pre;
    post(int,int)->(int,int,int) post;

    input -> pre;
    pre -> post -> output;
}
```

Code specifies that we are creating new operator called `process`. It takes one stream of integers as input and outputs one stream of triplets integers.

In the first part we declare sub operators, define type of input and output. For every declared sub operator we provide identifier. Second part specifies connec-

tion between declared operators. Code `op1 -> op2` indicates that output of `op1` is connected to input of operator `op2`. In this case output type of `op1` has equal to input type of `op2`. Bobolang syntax also allows to create chains of operators like `op1 -> op3` which has semantics like `op1 -> op2` and `op2 -> op3`.

There are explicitly defined operators called `input` and `output`. They represents input and output of declared operator `process`. The line `input -> pre;` represents that input of the operator `process` is connected to operator `pre`.

Boblang also allows to declare operators with empty input or output. They have type `()` that means it doesn't transfer any data. Only data allowed is to transfer poisoned pill. When box receives poisoned pill, it means that it should start working, Sending it means that it's work is done.

We can define whole execution plan using operator main with empty input and output. Example of whole Bobolang plan:

```
operator main()->()
{
    source()->(int) src;
    process(int)->(int,int,int) proc;
    sink(int,int,int)->() sink;

    input -> src -> proc -> sink -> output;
}
```

In figure 1.3 we can seen structure of example execution plan. Operators `init` and `term` are added automatically. Operator `init` sends poisoned pill to `source`, which can read data from hard drive or network. These data are send to box `process`. Operator sink stores data and sends poisoned pill to box `term` and the computation ends.

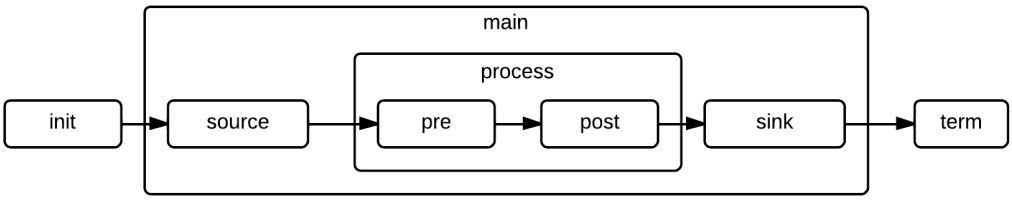


Figure 1.3: Example of execution plan.

1.3 Architecture

1.4 Goal

2. Related work

2.1 SQL and relational algebra

2.2 Optimization and plan genenration

3. Analysis

4. Implementation

Conclusion

Bibliography

[1] D. Bednárek, J. Dokulil, J. Yaghob, and F. Zavoral. *Bobox: Parallelization framework for data processing. Advances in Information Technology and Applied Computing*, 2012.

[2] Z. Falt, , D. Bednárek, K. Martin, J. Yaghob, and F. Zavoral. *Bobolang - a language for parallel streaming applications.* In 23rd international symposium on High-Performance Parallel and Distributed Computing. ACM, 2014.

[3] H. Garcia-Molina, , J. D. Ullman, J. Widom. *Database Systems The Complete Book.* Prentice Hall, 2002, ISBN 0-13-031995-3.

[4] Zbyněk Falt. *Parallel Processing of Data - Doctoral thesis.* Prague, 2013.

List of Tables

List of Abbreviations

Attachments