Stethoscope: A platform for interactive visual analysis of query execution plans

Mrunal Gawade
CWI, Amsterdam
mrunal.gawade@cwi.nl

Martin Kersten
CWI, Amsterdam
martin.kersten@cwi.nl

ABSTRACT

Searching for the performance bottleneck in an execution trace is an error prone and time consuming activity. Existing tools offer some comfort by providing a visual representation of trace for analysis. In this paper we present the Stethoscope, an interactive visual tool to inspect and analyze columnar database query performance, both online and offline. It's unique interactive animated interface capitalizes the large data-flow graph representation of a query execution plan, augmented with query execution trace information. We demonstrate features of Stethoscope for both online and offline analysis of long running queries. It helps in understanding where time goes, how optimizers perform, and how parallel processing on multi-core systems is exploited.

1. INTRODUCTION

Understanding database query execution traces is one of the most complex issues in database system research. Their analysis is needed to understand and to achieve optimal performance. Different implementations of database systems use different query representation formats. Queries vary in their complexity and so do their plans.

A query execution trace is a good starting point to reflect upon the run-time behavior. In offline mode, the execution steps taken can be inspected in detail for unanticipated behavior. In online mode, it can provide insight in the total system behavior. For example, influence of concurrent processes competing with the resources. Performance analysis tools are often specific to the DBMS, however the fundamental techniques used could be utilized, in the design of new tools.

In this demo, we present Stethoscope, a platform to analyse MonetDB [3] query plans and their execution traces. Each query plan models a dataflow dependency, which allows it to be represented as a directed acyclic graph (DAG), described in a dot file. Stethoscope combines dot file and execution trace to build a powerful tool, which animates the execution trace and provides navigational access to the portions of interest in the plan. This way, it can be used to monitor long running queries and performance bottlenecks in the kernel. Stethoscope provides the following features:

1. Interactive animated navigation in complex query plans.
2. Color coded monitoring of query execution state changes.
3. Run time analysis of execution states using debug window, tool tip text.
4. Flexible options for filtering of execution traces.
5. Support for large query plans with graph representation of more than 1000 nodes.

The interactive animated features by Stethoscope makes analysis more interesting and less error prone, as compared to other tools, which rely on manual browsing methods, in a static graphical representation of an execution trace. In the remainder of the paper, we provide a brief overview of the Stethoscope architecture, demonstration guidelines, and lessons learned during its implementation.

2. BACKGROUND

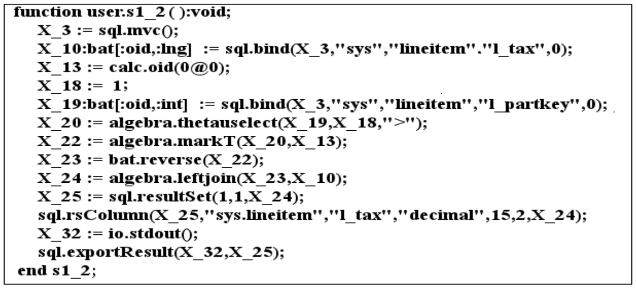
MonetDB is an open-source columnar database system developed at CWI [3]. It is predominantly used for OLAP based workloads. One of the prime features of MonetDB is its ability to support multiple front end data models, including query language interfaces such as SQL, SciQL, JSON/JAQL, and XQuery, etc. Client connections could be also made through PHP, JDBC, and ODBC interfaces. MonetDB uses the MonetDB Assembly Language (MAL), as an intermediate language to represent query plans. For example, a SQL query gets parsed and is converted into a relational algebra representation. This algebra representation is then converted to a MAL plan. Subsequently, optimizers work on the generated MAL plan to derive an optimized MAL plan. The final MAL plan is then interpreted. Figure 1 displays a MAL plan for the SQL query described next, from a TPC-H schema [6]. A graph representing complex MAL plan could grow very large in size. Figure 2 displays such a graph.

select l_tax from lineitem where l_partkey=1.

The plan is a sequence of semantically arranged MAL instructions. The literals starting with "X_" are MAL variables. MAL variables are assigned return values of MAL statements [9]. A MAL statement has a syntax of the form "module.function(variable1,variable2,...)". For example, in the statement "algebra.leftjoin(X_23,X_10)", "leftjoin" is a function in the "algebra" module. MAL comprises of a set of modules and a set of functions supported by each module.

MonetDB provides a GDB-like MAL debugger for runtime

Proceedings of the VLDB Endowment, Vol. X, No. Y



```
function user.s1_2 ( ):void;
    X_3 := sql.mvc();
    X_10:bat[:oid,:lng]  := sql.bind(X_3,"sys","lineitem"."l_tax",0);
    X_13 := calc.oid(0@0);
    X_18 :=  1;
    X_19:bat[:oid,:int]  := sql.bind(X_3,"sys","lineitem","l_partkey",0);
    X_20 := algebra.thetauselect(X_19,X_18,">");
    X_22 := algebra.markT(X_20,X_13);
    X_23 := bat.reverse(X_22);
    X_24 := algebra.leftjoin(X_23,X_10);
    X_25 := sql.resultSet(1,1,X_24);
    sql.rsColumn(X_25,"sys.lineitem","l_tax","decimal",15,2,X_24);
    X_32 := io.stdout();
    sql.exportResult(X_32,X_25);
 end s1_2;
```

Figure 1: MAL plan for a simple SQL query

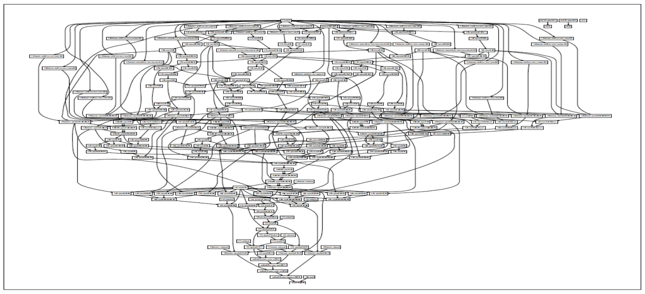


Figure 2: Large graph for a complex SQL query

inspection. However, further improvements could be gained by having a visual assistance tool that analyzes MAL execution traces. Stethoscope helps here by providing a set of functionalities to analyze MAL plans, in a fast and efficient manner.

3. SYSTEM ARCHITECTURE

The Stethoscope is a Java application and integrates open-source products such as the the MonetDB profiler [3], GraphViz library [4], and ZGrviewer component [7] of ZVTM toolset [8]. We describe each of these tools here.

Mserver is the MonetDB database server [3]. It is the main component which encapsulates the entire MonetDB execution environment. Mserver works as a background process. It listens for the incoming client connections on user defined ports. Stethoscope connects to Mserver as a client.

The MAL profiler is a component in MonetDB kernel which profiles executed MAL instructions. It supports profiling of events using several OS-specific properties, such as IO behavior, memory usage and cpu state, and MAL statement state. The profiler accepts filter options set through Stethoscope, which enables it to profile only a subset of event types. The events are either sent over a UDP stream back to the Stethoscope, or are dumped in a file, for offline analysis.

The MonetDB server generates a dot file representation for each MAL plan before execution begins. A dot file represents a graph and describes the grammar for the representation of nodes, and the association between nodes and edges [2]. GraphViz can convert a dot file to a graph structure representation [4]. Stethoscope uses this graph structure representation to setup different navigational strategies.

3.1 ZGrviewer

ZGrviewer is an open source tool from the ZVTM tool set which provides interactive navigation functionality in a graph structure [7]. The highlight of ZGrviewer is the zoom-able interface which allows keyboard and mouse scroll based navigation with zooming ability on individual nodes and edges in a graph. ZGrviewer comes with a plethora of features such as set of lenses viz. fish eye lens, etc. for visual interaction with graph nodes. ZGrviewer is implemented in Java and the Stethoscope uses ZGrviewer interfaces for interactive navigation in the graph structure. The Stethoscope code integrates with the ZGrviewer code base in a modular manner and provides interfaces for extensibility.

ZGrviewer stores graphics related meta-information in multiple structured object representations. Glyph is a structure representing a fundamental graphical object in ZGrviewer [1]. For example, consider a two node graph, with one undirected edge between them. Assume each node is represented with the shape of a circle and has a text label associated with it. ZGrviewer uses a glyph object each, to represent the shape, text, and edge. Thus for our example graph, ZGrviewer maintains following objects, shape (two objects), text (two objects), and edge (one object). Other important objects are a virtual space, which represents a canvas on which graphs are dawn and a camera object, which shows different views at different zoom levels, in a virtual space.

3.2 Textual Stethoscope

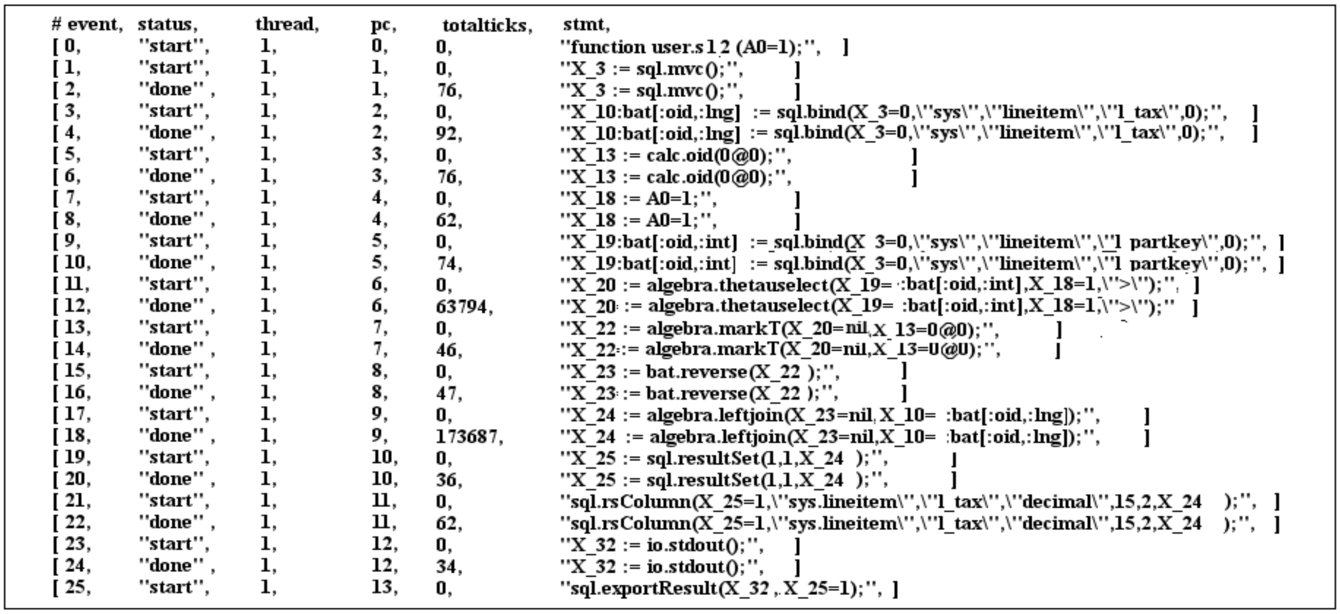
The MonetDB profiler information is accessed through a textual version of Stethoscope. It uses a UDP socket interface to connect to MonetDB server, for receiving the MonetDB execution trace. The textual Stethoscope can connect to multiple MonetDB servers at the same time to receive execution traces from all (distributed) sources. Its filter options allow for selective tracing of execution states on each of the connected servers. The profiler intercepted events on these servers are streamed back on a UDP connection to the textual Stethoscope. A sample execution trace from a trace file looks as given in the Figure 3.

3.3 Trace and Dot File Mapping

Each MAL instruction is represented in the trace in the Figure 3, with two events. A “start” event marks the start of the instruction and a “done” event marks the end of the instruction. The program counter (pc) is an important field in the trace, and is used to map pc to a node number in a dot file. For example, an instruction execution trace statement with pc=1 maps to the node “n1” in the dot file. The “stmt” field in instruction execution trace represents a MAL instruction and maps to the “label” field in the dot file.

4. WORK-FLOW DESCRIPTION

The Stethoscope works in both online and offline mode. Both modes share some fundamental steps, such as dot file parsing, conversion to an in memory graph representation, and sequential reading of a trace file. As a first step the dot file gets parsed and an intermediate scalar vector graphics (svg) representation gets created. In the next step, the svg file gets parsed and an in memory graph structure gets created. The root node of this graph structure is used to traverse the graph at a later stage. Both steps use the Graphviz library interface. As a next step, Stethoscope parses the trace file in a sequential manner, storing attributes of the trace file. The “event” attribute from the trace is used as



```
# event,  status,     thread,   pc,   totalticks,  stmt,
[ 0,      "start",    1,        0,    0,           "function user.s1 2 (A0=1);",   ]
[ 1,      "start",    1,        1,    0,           "X_3 := sql.mvc();",        ]
[ 2,      "done" ,    1,        1,    76,          "X_3 := sql.mvc();",        ]
[ 3,      "start",    1,        2,    0,           "X_10:bat[:oid,:lng]  := sql.bind(X_3=0,\"sys\",\"lineitem\",\"l_tax\",0);",    ]
[ 4,      "done" ,    1,        2,    92,          "X_10:bat[:oid,:lng]  := sql.bind(X_3=0,\"sys\",\"lineitem\",\"l_tax\",0);",    ]
[ 5,      "start",    1,        3,    0,           "X_13 := calc.oid(0@0);",               ]
[ 6,      "done" ,    1,        3,    76,          "X_13 := calc.oid(0@0);",               ]
[ 7,      "start",    1,        4,    0,           "X_18 := A0=1;",          ]
[ 8,      "done" ,    1,        4,    62,          "X_18 := A0=1;",          ]
[ 9,      "start",    1,        5,    0,           "X_19:bat[:oid,:int]  := sql.bind(X_3=0,\"sys\",\"lineitem\",\"l_partkey\",0);",  ]
[ 10,     "done" ,    1,        5,    74,          "X_19:bat[:oid,:int]  := sql.bind(X_3=0,\"sys\",\"lineitem\",\"l_partkey\",0);",  ]
[ 11,     "start",    1,        6,    0,           "X_20 := algebra.thetauselect(X_19= :bat[:oid,:int],X_18=1,\">\");",  ]
[ 12,     "done" ,    1,        6,    63794,       "X_20 := algebra.thetauselect(X_19= :bat[:oid,:int],X_18=1,\">\");"   ]
[ 13,     "start",    1,        7,    0,           "X_22 := algebra.markT(X_20=nil,X_13=0@0);",        ]
[ 14,     "done" ,    1,        7,    46,          "X_22 := algebra.markT(X_20=nil,X_13=0@0);",        ]
[ 15,     "start",    1,        8,    0,           "X_23 := bat.reverse(X_22 );",          ]
[ 16,     "done" ,    1,        8,    47,          "X_23 := bat.reverse(X_22 );",          ]
[ 17,     "start",    1,        9,    0,           "X_24 := algebra.leftjoin(X_23=nil,X_10= :bat[:oid,:lng]);",       ]
[ 18,     "done" ,    1,        9,    173687,      "X_24  := algebra.leftjoin(X_23=nil,X_10= :bat[:oid,:lng]);",       ]
[ 19,     "start",    1,        10,   0,           "X_25 := sql.resultSet(1,1,X_24  );",         ]
[ 20,     "done" ,    1,        10,   36,          "X_25 := sql.resultSet(1,1,X_24  );",         ]
[ 21,     "start",    1,        11,   0,           "sql.rsColumn(X_25=1,\"sys.lineitem\",\"l_tax\",\"decimal\",15,2,X_24   );",  ]
[ 22,     "done" ,    1,        11,   62,          "sql.rsColumn(X_25=1,\"sys.lineitem\",\"l_tax\",\"decimal\",15,2,X_24   );",  ]
[ 23,     "start",    1,        12,   0,           "X_32 := io.stdout();",     ]
[ 24,     "done" ,    1,        12,   34,          "X_32 := io.stdout();",     ]
[ 25,     "start",    1,        13,   0,           "sql.exportResult(X_32 ,.X_25=1);",  ]
```

Figure 3: MAL plan execution trace of a simple SQL query

an index to store the attribute contents. The "pc" attribute is mapped to a node name, to search for the corresponding node in the graph structure, during graph traversal.

4.1 Offline

The system uses event based programming interfaces to monitor click events and takes appropriate action in response. Prominent actions are navigate to the next node in the graph, change color of a node, and display tool-tip text. We describe the features related to these actions in the demonstration section. Offline mode needs access to a preexisting dot file and trace file. Once the off-line mode is selected, and the initial dot file parsing to graph structure creation stage is over, interactive analysis begins.

4.2 Online

In online mode, both dot and trace files are generated at run-time by MonetDB server. Online mode components use a multi-threaded design. As a first step, the textual Stethoscope is launched in a dedicated thread. The textual Stethoscope awaits in a listening mode for the arrival of trace stream on UDP connection. The trace received is redirected to a trace file.

The query whose execution plan needs to be analyzed is launched next in a separate thread. The trace file continuously receives the trace stream from the textual Stethoscope, while the query execution is in progress.

The dot file is a prerequisite for the graph structure generation. The MonetDB server generates the dot file content and sends it over on the UDP stream to the textual Stethoscope, before query execution begins. A separate thread monitors the received UDP stream for dot file and execution trace file content. It filters the dot file content, generates a new dot file, and stores the content in it.

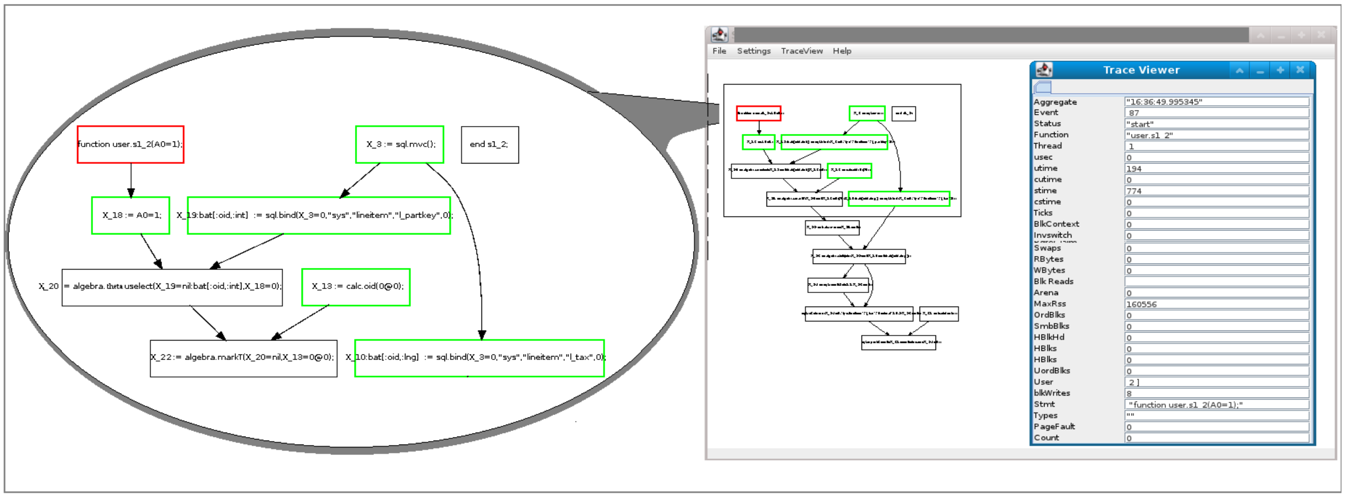
As the query execution begins, MonetDB profiler generates MAL instruction execution trace and sends to the textual Stethoscope. The monitoring thread filters the trace and a trace file is generated. As the trace file grows in size, its content is sampled in a buffer. MonetDB query plans have instructions where MAL operator takes long time to execute, for example a join operator. When this occurs, the execution trace blocks, resulting in blocking of the growth of trace file. An algorithm for run-time analysis, to filter lengthy MAL instructions is applied on the buffer content. We describe this algorithm in brief, in Section 4.2.1.

4.2.1 Run-time Analysis

Finding long running instructions in a MAL plan is one of the main purpose of the Stethoscope. Lengthy instructions could be filtered either on server or client side. They could be represented by color coding, progress window, and pop-ups. We focus on coloring of nodes to represent state change, on the client side.

Coloring graph nodes in an online stream is a complex task due to rendering limitations from the Java system. The Stethoscope uses the Java Event Dispatch thread queuing framework for queuing up nodes to render. This introduces a delay of up-to 150ms between rendering of consecutive nodes. A node is colored RED or GREEN based on the instruction status of "start" or "done" respectively.

Most instructions in the execution trace occur in sequence of pairs of "start" and "done" events. A consecutive "start" and "done" event status for the same instruction, with presence of more instructions afterwards, indicates that the instruction under analysis executed in least time. Hence, it is not a costly instruction. All such instructions are not colored. An instruction which does not appear in a sequence of pairs of "start" and "done" event is colored. For example, consider a pruned execution trace buffer from Figure 3, with fields {status,pc}, representing 6 instruction statements {start,1},{done,1},{start,2},{done,2},{start,3},{start, 4}. The graph nodes corresponding to first four statements will not be colored, as the two instructions corresponding to pc=1 and pc=2 occur in a pair, in a sequence. However,


function user.s1_2(A0=1);
X_3 := sql.mvc();
end s1_2;
X_18 := A0=1;
X_19:bat[:oid,:int] := sql.bind(X_3=0,"sys","lineitem","l_partkey",0);
X_20 = algebra.thetauselect(X_19=nil:bat[:oid,:int],X_18=0);
X_13 := calc.oid(0@0);
X_22 := algebra.markT(X_20=nil,X_13=0@0);
X_10:bat[:oid,:lng] := sql.bind(X_3=0,"sys","lineitem","l_tax",0);
File
Settings
TraceView
Help
Trace Viewer

Aggregate	"16:36:49.995345"
Event	87
Status	"start"
Function	"user.s1 2"
Thread	1
usec	0
utime	194
cutime	0
stime	774
cstime	0
Ticks	0
BlkContext	0
Invswitch	0
Swaps	0
RBytes	0
WBytes	0
Blk Reads	
Arena	0
MaxRss	160556
OrdBlks	0
SmbBlks	0
HBlkHd	0
HBlks	0
HBlks	0
UordBlks	0
User	2]
blkWrites	8
Stmt	"function user.s1 2(A0=1);"
Types	""
PageFault	0
Count	0

Figure 4: Display window for a simple MAL plan execution trace analysis

the graph node corresponding to the fifth instruction with pc=3 will be colored in RED. Hence, this graph node coloring algorithm doesn't check a specific instruction execution threshold time. We provide another algorithm which allows the user to specify an instruction execution threshold time.

5. DEMONSTRATION

The focus of the demonstration would be to exhibit various features of Stethoscope, while analyzing long running TPC-H queries in both online and offline mode.

Offline Demo: We use the trace replay feature to show utilization distribution of threads, memory usage by operators, and costly instruction clustering. A user can play with the following features, to analyze MAL plan execution trace.

- Step by step walk through, monitoring individual instruction using Stethoscope filter options window, debug options window, and tool-tip text display.
- Fast-forward, rewind, and pause functionality of the trace replay.
- Finding costly instructions by coloring during trace replay between two instruction states.
- Birds eye view of the entire trace, to understand the sequence of instruction execution clustering.
- Animation effects such as change of zoom level, color, and transition time between highlights of nodes.

Online Demo: Online mode exhibits similarity to the offline trace replay feature. Multi-core utilization analysis exhibits degree of multi-threaded parallelization of MAL instructions and interference from rest of the load on the system. Some other ways to analyze MAL plan execution are as follows.

- Monitor the progress of query plan execution, and highlight long running instructions based on the algorithm described in Section 4.2.1.
- Analyze runtime resource utilization by long running instructions using multiple instances of debug options window, and tool tip text display.

To illustrate, using Stethoscope we have uncovered several unusual cases, such as sequential execution of a MAL plan where multithreaded execution was expected.

6. CONCLUSION

Stethoscope is an extensible platform for query execution analysis in columnar systems such as MonetDB. The main contribution is a unique interactive animated interface for analysis of execution trace and the role it plays to gain performance insight. Developing an interactive visual front end platform using open-source tools has been a challenging task. We spent considerable time in understanding the code base of ZGrviewer, and experimenting to find the correct logic for coloring of individual graph objects, in online and offline mode. Rendering of nodes in tune with the online trace flow is a challenge, due to refresh rate limitations of the system. User interface and event based programming in general involves tedious manual testing approach.

We have the following features planned for a future release of the stethoscope. An analytic interface for micro analysis of trace, gradient coloring of graph nodes to display a range of execution times, and selective pruning of MAL plan to remove unimportant administrative instructions.

7. REFERENCES

[1] E. Pietriga. A toolkit for addressing HCI issues in visual language environments. *In VLHCC*, 2005.

[2] E. Gansner, E. Koutsoos, and S. North. Drawing graphs with dot. 2006.

[3] MonetDB, www.monetdb.org.

[4] GraphViz, www.graphviz.org.

[5] DOT, www.graphviz.org/doc/info/lang.html.

[6] TPC-H Benchmark, www.tpc.org/tpch.

[7] ZGrviewer, www.zvtm.sourceforge.net/zgrviewer.html.

[8] ZVTM, www.zvtm.sourceforge.net.

[9] MAL, www.monetdb.org/Documentation/Manuals/MonetDB/MALreference.