User Manual

GCX - Garbage Collected XQuery

Saarland University Database Group
Christoph Koch, Stefanie Scherzinger, Michael Schmidt

January 26, 2007

Contents

1 Preface

The GCX XQuery engine is the first streaming XQuery engine which implements active garbage collection [2], a novel buffer management strategy in which both static and dynamic analysis are exploited. This technique actively purges main memory buffers at runtime based on the current status of query evaluation. The approach aims at both keeping main memory consumption low at runtime and speeding up query evaluation. Please visit the project homepage

`http://www.infosys.uni-sb.de/projects/streams/gcx`

for detailed information on active garbage collection in XQuery engines.

2 Installation

2.1 Requirements

The easiest way to get started with GCX is to use the precompiled binaries at the download page. Currently, binaries for Linux und Windows (i386 architecture) are available.

If your system is not supported, you need to compile the GCX engine from sources. The following tools are required for manual compilation.

- GNU Make (installation tested with version 3.81beta4)
- GNU Bison (installation tested with version 2.1)
- GNU Flex (installation tested with version 2.5.31)
- GNU Sed (installation tested with version 4.1.4)

We provide Makefiles for both Windows and Linux.

2.2 Using the Binaries

If you choose to download the binaries, no installation will be necessary. Simply download the binary bundle that corresponds to your operating system (either *gcx_1-0beta_bin_win32.zip* or *gcx_1-0beta_bin_linux.tar.gz*) and unzip the file. This will create the directory structure

```
gcx_1-0beta_bin_OS
    examples
        dblp
        nasa
        xmark
        xmp
    bin
```

The executable (either `gcx` or `gcx.exe`, depending on whether you use the Linux or the Windows version) will be available in the `bin` directory. Simply open a shell, change into the `bin` directory, and run the executable. For a list of available command line options see Section 4.

2.3 Compiling the Sources

2.3.1 Compilation under Linux

1. Download the archive "*gcx_1-0beta_src.tar.gz*" (you can also download the .zip version of the archive) from

   `http://www.infosys.uni-sb.de/projects/streams/gcx`

2. Unpack the archive by typing
   `> tar -xzf gcx_1-0beta_src.tar.gz`

   This will create the directory structure

   ```
   gcx_1-0beta_src
       bin
       examples
           dblp
           nasa
           xmark
           xmp
       src
   ```

3. Enter the *src* directory by typing
   `> cd ./gcx_1-0beta_src/src`

4. Simply type
   `> make -f Makefile.Linux`
   to compile the sources. A binary file *gcx* will be created in the *bin* directory.

5. You might consider adding the *bin* directory to your `PATH` variable or creating a link in */usr/bin*.

2.3.2 Compilation under Windows

We propose to use the MinGW environment (http://www.mingw.org/) to install GCX under Windows. You will additionally need to install the following tools:

- GnuWin32 Make (http://gnuwin32.sourceforge.net/packages/make.htm)
- GnuWin32 Bison (http://gnuwin32.sourceforge.net/packages/bison.htm)
- GnuWin32 Flex (http://gnuwin32.sourceforge.net/packages/flex.htm)
- GnuWin32 Sed (http://gnuwin32.sourceforge.net/packages/sed.htm)

Unzip the archive and follow the installation instructions given in Section 2.3.1. When building the sources (step 4), type

```
> make -f Makefile.Windows
```

instead.

2.4 Compiling with Statistics Support

To compile a version that enables for debug printing buffer statistics, simply uncomment the line

```
# FLAGS = -DGENEREATE_STATS
```

in the Makefile (either *Makefile.Linux* or *Makefile.Windows*) in the *src* directory and type
`> make -f <Makefile> clean all`
to rebuild your application, where <Makefile> is either *Makefile.Linux* or *Makefile.Windows*.

Warning. *Compiling the application with support for statistics might significantly slow down query evaluation and is not a recommended compile option!*

We refer the reader to Section 4.1.14 for a description of the corresponding command line option that can be used to query the buffer statistics.

3 Supported Query Language

GCX supports composition-free XQuery [1], i.e. without let-expressions. GCX allows nested for-loops, child- and descendant axis in path step expressions, if-then-else statements, and compares nodes by their string values. Currently, all path step expressions are single steps, i.e. `for $x in $y/a` is supported while `for $x in $y/a//b` is not. Please note that in most practical cases, multi-step for loops can easily be rewritten into single loops. An explicit grammar of the XQuery fragment supported is provided in Figure 1.

4 Command-Line Arguments

Usage: `gcx [OPTIONS] <query_file> <xml_file>`

For instance, you can simply run a query from file *query.xq* against document *doc.xml* by typing

`> ./gcx query.xq doc.xml` (on Linux)

or

`> gcx.exe query.xq doc.xml` (on Windows).

4.1 Debug Options

The following command line debug options are available.

4.1.1 `--pfileinfo`

Prints query input file and XML file.

$$
\begin{aligned}
\mathit{XQuery} &::= \epsilon \mid \mathit{XMLExpr} \\
\mathit{XMLExpr} &::= \langle \mathit{QName} \rangle\ \mathit{NestedXMLExpr}\ \langle / \mathit{QName} \rangle \mid \langle \mathit{QName} \rangle \langle / \mathit{QName} \rangle \\
\mathit{NestedXMLExpr} &::= \{\, \mathit{QExpr} \,\} \mid \mathit{String} \mid \mathit{XMLExpr} \mid \mathit{NestedXMLExpr}\ \mathit{NestedXMLExpr} \\
\mathit{QExpr} &::= \mathit{ReturnQExpr} \mid \mathit{QExpr}, \mathit{QExpr} \\
\mathit{ReturnQExpr} &::= \mathit{QExprSingle} \mid (\mathit{QExpr}) \mid () \\
\mathit{QExprSingle} &::= \text{"}\mathit{String}\text{"} \mid \mathit{FWRExpr} \mid \mathit{IfExpr} \mid \mathit{VarExpr} \mid \mathit{NestedXMLExpr} \\
\mathit{FWRExpr} &::= \mathit{ForClause}\ [\textbf{where}\ \mathit{Condition}]^{?}\ \textbf{return}\ \mathit{ReturnQExpr} \\
\mathit{ForClause} &::= \textbf{for}\ \$\mathit{VarName}\ \textbf{in}\ \mathit{VarAxisExpr}\ [,\ \$\mathit{VarName}\ \textbf{in}\ \mathit{VarAxisExpr}]^{*} \\
\mathit{IfExpr} &::= \textbf{if}\ (\mathit{Condition})\ \textbf{then}\ \mathit{ReturnQExpr}\ \textbf{else}\ \mathit{ReturnQExpr} \\
\mathit{Condition} &::= \textbf{fn:true()} \mid \textbf{fn:false()} \mid \textbf{fn:exists(}\mathit{VarAxisExpr}\textbf{)} \\
&\quad \mid (\mathit{Condition}) \mid \mathit{Operand}\ \mathit{RelOp}\ \mathit{Operand} \mid \textbf{fn:not(}\mathit{Condition}\textbf{)} \\
&\quad \mid \mathit{Condition}\ \textbf{and}\ \mathit{Condition} \mid \mathit{Condition}\ \textbf{or}\ \mathit{Condition} \\
\mathit{Operand} &::= \mathit{VarExpr} \mid \text{"}\mathit{String}\text{"} \\
\mathit{RelOp} &::= < \mid <= \mid >= \mid > \mid = \mid != \\
\mathit{VarExpr} &::= \$\mathit{VarName} \mid \mathit{VarAxisExpr} \\
\mathit{VarAxisExpr} &::= \$\mathit{VarName}\ \mathit{PathStepExpr} \mid \mathit{PathStepExpr} \\
\mathit{PathStepExpr} &::= / \mid (/) \mid \mathit{Axis}\ \mathit{NodeTest} \\
\mathit{Axis} &::= / \mid \textbf{/child::} \mid // \mid \textbf{/descendant::} \\
\mathit{NodeTest} &::= \textbf{node()} \mid \textbf{text()} \mid * \mid \mathit{QName}
\end{aligned}
$$

Figure 1: The XQuery fragment supported in GCX where *VarName* denotes a variable name, *QName* a tagname, and *String* stands for a string constant.

4.1.2 `--pqueryorig`

Prints the parsed query as read by the parser.

4.1.3 `--pqueryopt`

Prints the query after the optimization phase.

4.1.4 `--pqueryrew`

Prints the rewritten query with *signOff* statements.

4.1.5 `--pprojtree`

Prints the constructed base projection tree.

4.1.6 `--pprojnfa`

Prints the projection NFA.

4.1.7 `--pprojdfa`

Prints the initial projection DFA.

4.1.8 `--pfsamap`

Prints the mapping from variables to their first straight ancestors.

4.1.9 `--ppathenv`

Prints the path environment as computed during role computation.

4.1.10 `--pinitbuf`

Prints the initial empty buffer.

4.1.11 `--pfinalbuf`

Prints the buffer at the end of query evaluation. After correct evaluation of a query, the final buffer should be empty.

4.1.12 `--pfinaldfa`

Prints the final projection DFA. The projection DFA is constructed lazily during query evaluation and depends on both the query and the XML input document.

4.1.13 `--ptagmap`

Prints the mapping integers to tag names.

4.1.14 `--pstats`

Prints evaluation statistics, e.g. the maximal number of buffered tokens. Note that this option is only available if the application has been compiled with the `-DGENERATE_STATS` option (see Section 2.4).

4.1.15 `--pdebug`

Prints all debug options described in Sections 4.1.1 through 4.1.14.

4.2 Other Command-Line Options

4.2.1 `--version`

Prints the version number.

4.2.2 `--streamnodeb`

Prints the projected XML stream. Note that stream preprojection as a stand-alone tool is currently not implemented in a memory efficient way, i.e. the whole stream is loaded into the buffer before it is output.

4.2.3 `--stream`

Prints the projected XML stream together with additional debug information (in particular with associated roles). Stream preprojection as a stand-alone tool is currently not implemented in a memory efficient way, i.e. in this mode the whole stream is loaded into the buffer before it is output.

5 Sample Queries

The distribution contains a set of test queries together with matching XML documents. The test queries can be found in the *examples* directory. In the subdirectory there is a set of queries and an XML document matching to these queries.

6 Contact

For questions, bug reports and feature requests please use the GCX Mailing list at

`http://lists.math.uni-sb.de/mailman/listinfo/gcx-users`

or contact

`scherzinger@infosys.uni-sb.de`

or

`schmidt@infosys.uni-sb.de`

References

[1] C. Koch. "On the complexity of nonrecursive XQuery and functional query languages on complex values". *ACM Transactions on Database Systems*, 31(4), 2006.

[2] M. Schmidt, S. Scherzinger, and C. Koch. "Combined Static and Dynamic Analysis for Effective Buffer Minimization in Streaming XQuery Evaluation". To appear in Proc. ICDE 2007.