XML path matching for query processing

Guoquan Lee
Faculty Advisor: Dr. Zachary G. Ives

TECHNICAL APPROACH

Modifications were made to the existing x-scan algorithm to handle XQuery expressions with \texttt{//}.

OVERVIEW OF EXISTING X-SCAN ALGORITHM

Refer to Figure 1 (Figures 1, 2, and 3 are reproduced with permission from Ives (2002)). The XML stream is processed by a SAX parser. The XML data is stored in the XML Tree manager and is also matched against finite state machines that are responsible for XPath matching. X-scan converts each XPath into an equivalent DFA.

Figure 1: x-scan process

XML Stream
\begin{itemize}
  \item \texttt{<xml>}
  \item \texttt{<book>}
  \item \texttt{<title>}
  \item \texttt{<publisher>}
  \item \texttt{<author>}
  \item \texttt{</book>}
\end{itemize}

Figure 2: XQuery Example

\begin{verbatim}
$result =
  \texttt{FOR} \texttt{$x} \texttt{IN} \texttt{document("books.xml")/db/book},
  \texttt{$t} \texttt{IN} \texttt{$x/title/data()},
  \texttt{$a} \texttt{IN} \texttt{$x/author/data()}
  \texttt{RETURN <item>}
  \texttt{<person>{\$n}</person>}
  \texttt{<pub>{\$t}}</pub>}
\end{verbatim}

Figure 3: State machines corresponding to XPaths in Figure 2

THE EXTENDED X-SCAN ALGORITHM

\begin{itemize}
  \item We simulated multiple instances of each DFA running in parallel for each SAX event received.
  \item Each DFA has a thread table entry to keep track of each of its threads. Each thread is linked to an object (DFAManager) whose main purpose is to write into the Tree Manager and the output buffer.
\end{itemize}

Figure 4: Collaboration diagram of the extended x-scan process when a start tag is received

Memory Management

\begin{itemize}
  \item Memory management is challenging as the XML files we are processing are a few hundred MB.
  \item Electric Fence and GDB debugger were used for memory management.
\end{itemize}

Pseudo Producer-Consumer problem

\begin{itemize}
  \item Modifications to algorithms that solve the producer-consumer were applied to serve X-scan’s purposes.
\end{itemize}

Extending x-scan to handle \texttt{//}

\begin{itemize}
  \item To simulate “multi-threading”, we think of the DFA as a process which acts to group resources together. Each instance of the DFA running during XML path matching is considered to be a thread.
  \item Referring to Figure 4, when an XML Parser encounters an event, it will call an event handler of the object which manages all the DFAs (DFAManager).
  \item The DFAManager will iterate through its list of active DFA objects and notify each “thread” of the DFA which proceeds to do XML path matching independently of the other threads.
\end{itemize}

PRINCIPAL TECHNICAL CHALLENGES FACED

\begin{itemize}
  \item We built a component of a query processing system
  \item We learned to take a theoretical Computer Science problem and applied its solutions to solve the task at hand.
  \item Reducing duplication and unnecessary object creation could be a massive boast to performance.
\end{itemize}

CONCLUSION

\begin{itemize}
  \item \texttt{grep} was used to identify methods to be rewritten.
  \item Optimizations applied:
    \begin{itemize}
      \item efficient data structures (Hash tables)
      \item “Tokenization”
      \item removed unnecessary object creation.
    \end{itemize}
  \item Performance improved by about 4 times. Results attained by the x-scan operator on a 1.4GHz Linux machine are given in Table 1.
  \item The XQuery expressions used for the timings were of the form:
    \begin{verbatim}
    for $x in document("a.xml")/b/c,
    $y in $x/d
    \end{verbatim}
    Note that \texttt{a.xml} denotes the xml file that we are parsing; \texttt{b}, \texttt{c}, and \texttt{d} are element names.
\end{itemize}

Table 1: Timing Results

<table>
<thead>
<tr>
<th>File Size (MB)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>14.02s</td>
</tr>
<tr>
<td>5</td>
<td>39.20s</td>
</tr>
<tr>
<td>10</td>
<td>1.05s</td>
</tr>
<tr>
<td>50</td>
<td>3.06s</td>
</tr>
<tr>
<td>100</td>
<td>5.09s</td>
</tr>
<tr>
<td>250</td>
<td>8.05s</td>
</tr>
<tr>
<td>500</td>
<td>15.02s</td>
</tr>
<tr>
<td>1000</td>
<td>39.33s</td>
</tr>
</tbody>
</table>

CIS Department, University of Pennsylvania
In this project, we extended and optimized the x-scan operator in the Tukwila system to handle XQuery expressions with descendant-or-self axis (///). Modifications were made to the existing x-scan algorithm to handle XQuery expressions with //.

Refer to Figure 1 (Figures 1, 2, and 3 are reproduced with permission from Ives (2002)). The XML stream is processed by a SAX parser. The XML data is stored in the XML Tree manager and is also matched against finite state machines that are responsible for XPath matching. X-scan converts each XPath into an equivalent DFA.

The XML stream is processed by a SAX parser. The XML data is stored in the XML Tree manager and is also matched against finite state machines that are responsible for XPath matching. X-scan converts each XPath into an equivalent DFA.

We simulated multiple instances of each DFA running in parallel for each SAX event received. Each DFA has a thread table entry to keep track of each of its threads. Each thread is linked to an object (IOManager) whose main purpose is to write into the Tree Manager and the output buffer.

Optimization of extended x-scan
- gprof was used to identify methods to be rewritten.
- Optimizations applied:
  - efficient data structures (Hash tables)
  - “Tokenization”
  - removed unnecessary object creation.
- Performance improved by about 4 times. Results attained by the x-scan operator on a 1.4GHz Linux machine are given in Table 1.

For $x$ in document("a.xml")/b/c, $y$ in $x$/d
Note that $a.xml$ denotes the xml file that we are parsing; $b$, $c$, and $d$ are element names.

Memory management
- Memory management is challenging as the XML files we are processing are a few hundred MB.
- Electric Fence and GDB debugger were used for memory management.

Pseudo Producer-Consumer problem
- Modifications to algorithms that solve the producer-consumer were applied to serve x-scan’s purposes.

Extending x-scan to handle //
- To simulate “multi-threading”, we think of the DFA as a process which acts to group resources together. Each instance of the DFA running during XML path matching is considered to be a thread.
- Referring to Figure 4, when an XML Parser encounters an event, it will call an event handler of the object which manages all the DFAs (DFAManager).
- The DFAManager will iterate through its list of active DFA objects and notify each “thread” of the DFA which proceeds to do XML path matching independently of the other threads.

We built a component of a query processing system
- We learned to take a theoretical Computer Science problem and apply its solutions to solve the task at hand.
- Reducing duplication and unnecessary object creation could be a massive boast to performance.