KNSP: A Kweelt - Niagara based Quilt Processor Inside Cocoon over Apache

Xidong Wang & Shiliang Hu
{wxd, shiliang}@cs.wisc.edu
Department of Computer Science, University of Wisconsin Madison

1. Introduction

To study the exact operation process of web publishing systems, we installed and studied the Apache Web server, Jakarta-Tomcat Servlet Engine, Cocoon web publishing framework and the Kweelt Quilt processing mechanism. We partially replaced KSP (Kweelt in Cocoon) processor with Niagara 1.0 and realized an interesting KNSP, Kweelt and Niagara based Server Page processing scheme.

Section 2 introduces the Cocoon web publishing framework and relevant background on the Apache web server, Jakarta-tomcat Servlet Engine and two important XML Query languages – Quilt and XML-QL. Section 3 introduces the architecture and implementation of our KNSP, Kweelt-Niagara based Server Page system. Preliminary performance results and comparisons are presented in Sections 4. We conclude in section 5.

2. Cocoon Web Publishing Framework

Cocoon is a web publishing framework that relies on new W3C Technologies (such as DOM, XML, and XSL) to provide web content. The Cocoon project aims to change the way web information is created, rendered and served. The new Cocoon paradigm is based on the fact that document content, style and logic are often created by different individuals or working groups. Cocoon aims for a complete separation of the three layers, allowing the three layers to be independently designed, created and managed, reducing management overhead, increasing work reuse and reducing time to market.

As part of the Apache XML project, the cocoon web publishing system embeds a Quilt query processor called Kweelt inside Apache web server so that the web server can run queries on XML data sources.

A rough image about the operation scenario of an XML-support web server, say the Apache web server with Cocoon inside, may look like this way: First, some sort of client, say a web browser or wireless palm, sends its requests in URI or POST form to a web server. The server routes this request to an appropriate processor or engine inside itself based on the URI request. For XML-related request, it will be dispatched to the Cocoon system, which functions at an engine level, or in other words it manages various actors such all kinds of producers, processors, formatters, interpreters or transformers. The request will be directed by Cocoon Engine to an appropriate producer to generate XML document, possibly embedded with Quilt queries or other kinds of embedded logic. This XML document will be further processed by various processors that the Cocoon system has based on the kinds of logic embedded in the XML document. This processing keeps until no further server end processing and including is needed. Each processor may process lots of XML data sources during this process to replace logic with content.
Then the result XML file, which has the user desired content, will be formatted with appropriate formatters, based on the desired style or the type of the user-agent, to certain formats, such as html file, PDF document or WML for mobile devices. Finally, the well-styled document for the client is sent back to the happy client. We talk briefly about the Apache web server and Jakarta-Tomcat Servlet Engine in §2.1, Cocoon publishing system in §2.2. We also introduce XML-QL and Quilt in §2.3 as the background of our NSP efforts.

### 2.1 Apache Web Server & Jakarta-Tomcat Servlet Engine

When Apache starts, it loads the jserv module mod_jserv.so into its space, then spawns one main process and multiple (default 5) worker processes to serve client’s request. Main process will listen for HTTP requests. When client send a HTTP request, main process will place the available socket onto a list of jobs. Then any one of worker process will take the top job off the job list and handle the connection to completion. If the request should be processed by Jserv engine, worker process makes use of mod_jserv.so, which is provided by Jserv engine, to connect with Jserv. Special protocol is designed and implemented for sending parameter and receiving answer.

![Figure 1: apache and Jserv](image)

### 2.2 The Framework of Cocoon Web Publishing System

We use the module invocation relationship graph in Figure 1 to capture the framework of Cocoon web publishing system. As mentioned in §2.1, http requests are delivered from Apache processes to Jakarta-Tomcat Jserv through the special communication protocol. Jakarta TCP Listener thread accepts http request connections and spawn an **TcpConnectionThread** to handle the request. Most of the modules in Figure 2 fit into this magic thread for the request processing.

The leftmost module in Figure 2 stands for the **TcpConnectionThread** that runs the Jakarta-Tomcat Servlet Engine code. It uses **HttpConnectionHandler** to have **ContextManager** to deliver requests to the right Servlet processor wrapped in **ServletWrapper** interface. In our case, the request is to be delivered to the Cocoon publishing module. However, requests could also be routed to other Servlets. For example, JSP requests, which have URLs ended with *.jsp, are routed to Jasper JSP servlet.

Cocoon is a processor engine in the sense that it manages various kinds of producers, processors, formatters, interpreters, transformers, cache managers etc. Producers generate XML document based on http request. Processors are invoked based the processing instructions embedded within the XML document and process the corresponding type of logic in the document and replace them with processing results. In this project, we are only interested in one kind of logic – Quilt Queries embedded inside the
XML document. Therefore, we talk in detail about the Kweelt Quilt Processor in detail because it is the only Quilt Processor available for Cocoon by now. Formatters are used to render desired document style, which is out of our interests for now, so are other actors, like interpreter and transformer.

**Figure 2. Module Invocation Relationships**

Cocoon process Quilt logic in XML document by **KweeltProcessor** Quilt processor, which parses the Quilt Query with a parser generated by JavaCC. As we will see later that the parser specification file for JavaCC is main battlefield in this project. **KweeltProcessor** parsed Quilt query is in Kweelt’s AST and will be evaluated recursively without generating a logical plan and optimizing the query plan. The functional programming style recursive evaluation mechanism is pretty clear and neat, but could be very inefficient. We partially replaced Kweelt Quilt processor in Cocoon with Niagara version 1.

**Figure 3. Processes/Threads Relationships**

Figure 3 show the process and thread relationship in the total architecture. Apache web server, front end of total system, is a bunch of process, in which one is main process for listening for requests and
others are worker process. Jserv as a whole is one process with many threads. Some of threads are Jserv thread for apache requests, others are Niagara operator threads started by KNSP. We can see from Figure 3 that Jserv, Cocoon, AST part of KSP and Niagara query engine part are resident in one thread, TcpConnectionThread. Niagara search engine is another process to provide some assistance for query engine.

2.3 XML Query Languages : XML-QL and Quilt, What different?

According to current trend, there will be huge amount of web-accessible XML-files in the near future. How to query this up-coming World Wide Database becomes interesting research and development issues. There are many proposed XML query languages, for example, XML-QL, XQL and Quilt.

One of the most important issues in these query languages is the navigation mechanism for the nested structure of the XML files.

XML-QL was one of the earliest XML query language. A typical XML-QL query consists of a WHERE clause to specify data source and query conditions and a CONSTRUCT clause to specify the XML elements to be returned for the query. XML-QL uses patterns, similar to the XML element structure to specify both the navigation of query conditions and the pattern of query results. XML-QL supports wildcards to query XML files across the Internet.

On the other hand, the new Quilt query language adopted the XPATH navigation method. To some extent, XPATH looks similar to the normal navigation method in a typical file system by specifying the next navigation direction and desired components. However, XPATH is much more flexible. Currently, Quilt doesn’t support wildcards to query across the Internet.

Although Quilt borrows idea from many languages, its conceptual integrity comes from a deep reliance on the structure of XML, which is based on hierarchy, sequence, and reference. Quilt is able to express queries based on document structure and to produce query results that either preserve the original document structure or generate a new structure. Quilt can operate on flat structures, such as rows from relational databases, and generate hierarchies based on the information contained in these structures. It can also express queries based on parent/child relationships or document sequence, and can preserve these relationships or generate new ones in the output document. Although Quilt can be used to retrieve data from objects, relational databases, or other non-XML sources, this data must be expressed in an XML view, and Quilt relies solely on the structure of the XML view in its data model.

Quilt is an XML query language that not only conforms to the W3C XML query language requirements, but also combines a lot of advantages of other existing XML query languages, hence, provides more flexibility and capability. Many people believe Quilt could become the standard of XML query language instead of XML-QL.

3. NSP Architecture and Implementation

Our NSP processing is totally embedded inside the Cocoon framework. The only change we made is replacing the Kweelt Quilt query evaluation with the Niagara query processing.

3.1 Data Structure in Kweelt and Niagara

Cocoon passes XML documents with Quilt queries to KSP processor, KweeltProcessor. KweeltProcessor parses XML documents with certain XML parsers such Sun, IBM or Xerces parser, and extracts kweelt-query nodes out of the documents and processes them one by one. For each of the
**kweelt-query** node. **KweeltProcessor** calls **QuiltParser** to parse the query and generate the Quilt Abstract Syntax Tree. Originally KSP process the Quilt AST by evaluate it recursively.

While more efforts are taken in Niagara in order to promote parallelism and improve performance. Logical plan is produced from Abstract Syntax Tree. Query optimization techniques are taken on logical plan to get physical plan. The physical operators in final physical plan are executed in parallel to achieve high performance.

In order to get high performance by integrating Quilt parsing facility of Kweelt and Query optimization and evaluation advantage of Niagara, data structures in both sides must be mapped and integrated smoothly. Since there is no logical plan or physical plan in Kweelt, KNSP is designed to map AST of Kweelt to AST of Niagara, then all work afterwards such as generating logical plan form AST and optimizing logical plan to physical plan, are implemented in Niagara.

![Diagram of Kweelt processor](attachment:image.png)

Figure above shows that most of our work is mapping program whose input is AST in Kweelt and output is AST in Niagara. After that, instead of letting KSP to process it further, we pass the query containing AST of Niagara to the Niagara query engine for further processing. The Niagara query engine returns a stream of the query result. We replace the corresponding kweelt-query node with its query results for the content of the requested XML file. After all processing finish, we return the processed XML file back to the Cocoon Engine. Cocoon will process the XML file further and/or format it, e.g. with XSLT, and return the proper result media to the client to finish its URI request.

### 3.2 The Mapping between Kweelt AST & Niagara AST

Kweelt parse the quilt query into a Syntax tree and this is realized by parsing quilt query recursively with Quilt Language Grammar, constructing one expression node for each step. Therefore those expression nodes in different level of recursion form a syntax tree. The type or content of expression node in syntax tree do not catch our attention, but the construction procedure of syntax tree really do since it reveal the parse structure of that query. The construction procedure is realized in **QuiltParser.jj**, a java parser specification document to be compiled by JavaCC package. Our work is to insert some Niagara related codes into some steps in syntax tree construction procedure, utilizing its parse procedure to produce Niagara syntax tree. **We do not map directly between Kweelt AST and Niagara AST, instead we make use of the construction procedure of Kweelt AST to construct Niagara AST.**

The core of an XML-QL query consists of a WHERE clause that binds one or more variables, and a CONSTRUCT clause that uses the values of the bound variables to construct a new document. From XML-QL, Quilt borrows the concept of constructing a new document from bound variables, but Quilt uses a different paradigm for binding the variables and a richer data model to represent the result of the binding.

In Niagara query about one document or one kind of documents is encapsulated in a inClause structure, whose source element indicates the source of documents to be evaluated, dtidType element indicates the document type evaluated documents should conform to, and pattern element points to a pattern tree preserving the information about hierarchic and sequential relationships among the elements of the evaluated document. In syntax tree construction procedure step of Kweelt, we have to build all these data structures of Niagara. The recursive steps used to build Kweelt syntax tree is carefully studied and
some codes are injected. In the bottom level step of Kweelt to build document expression node, Niagara
inclause object is built and its dtdType and source element are assigned with information from Kweelt
document expression node. When Kweelt parse Quilt’s powerful XPATH expression, hierarchical
structures of pattern tree in inClause object of Niagara is set up. When Kweelt make use For or Let clause
to bind some variables, some nodes in pattern tree in inClause object of Niagara save this binding to its
data members. Such kind of inClause construction happens when Kweelt build construction expressions to
return result, since those return clauses also include some XPATH expressions.

One thing we would like to mention is that Quilt is more powerful than XML-QL, so it is great
possible that one Quilt query cannot be mapped to an equivalent XML-QL query. This belongs to
difference of query language. Since this project mainly deals with feasibility of integration of Kweelt and
Niagara and insert the result of integration into Cocoon as a processor, and demonstration of Niagara’s
performance advantage, we ignore the possible difference of two languages.

3.3 Kweelt & Niagara Integration.

After work on QuiltParser.jj parser specification file, the Quilt query parser generated by
JavaCC will return queries conform to the query data structure in the Niagara system. We added another
methods, executeQuiltQuery to the Niagara QueryEngine module. This method calls logical plan
generator in Niagara to generate logical plan and then calls Niagara query optimizer to optimize the logical
plan and generate the physical plan. The physical plan is submitted for execution by appending the plan to
the query queue. After this point, the execution of the Quilt query in NSP is identical to the normal query
processing in Niagara system. The query is scheduled by execution scheduler, among physical operator
queues, and the execution scheduler also builds the streams that connect these physical operators so that
they can be processed in the pipeline style to enhance efficiency. Physical operators are executed by
physical operator threads.

4. Preliminary Performance and Structure Evaluation

Query evaluation in Kweelt is very primitive. There is an eval function in each node of syntax tree
and evaluation is executed directly in syntax tree recursively in top to bottom order. Since document
objects can only be obtained in document expression node of bottom level, in fact actual evaluation work
are executed in bottom to top order and results are returned in same way. No query optimization are
taken and query is executed in single thread way. All intermediate results are kept in an environment
object transferred in eval functions of different levels. When join operation is executed, tuples from one
document are kept in environment, and for each tuple, query for the other documents are executed and that
tuple are joined with result tuples of the second document. So it is actually a nested loop join. And since
query for the second documents are executed multiple times, its performance will much lower than nested
loop join, in which query for two documents are executed once and results are joined. And there is no
optimization work in Kweelt. We can easily write a query in which there are scan and join related to one
document and Kweelt will execute join first and scan next. If selection factor of scan is small, Kweelt
will lose in performance again. Since Niagara is designed and implemented in parallel way, we can
conclude that Niagara’s performance must be much better than Kweelt, so KNSP will enjoy much better
performance than KSP.

Environment set up and result data............................................
5. Conclusions and Future Work

References

1. Java and XML By Brett McLaughlin 1st Edition June 2000
4. The Niagara Internet Query System. Jeffrey Naughton, David DeWitt, David Maier et. al..
5. Kweelt: http://db.cis.upenn.edu/Kweelt
7. XML, XSL etc: http://www.w3.org