

XML Indexing

XML Needs More than XML Indices

Effectively and efficiently querying a collection of XML documents is a challenge for several reasons. An XML document has a hierarchical, potentially recursive, structure without any predefined limit to the number of levels. Within a given collection the structure of the documents may vary from one to another in both the depth and structure of the hierarchy. The XML model has sometimes been called “semi-structured” to highlight the fact that in the same document structured data are freely mixed with textual information, and that content and meta data are conjoined in order to be machine readable (and, perhaps, to even be read by humans).

While XML standards are important they don't simplify the complexity of actual documents or make it easier to master their granular manipulation. Just by looking at two examples of standards in very different areas makes clear that the potential of XML is not reduced by any business schema. In the attached annex, the structure of FPML (the XML standard used in financial applications to process financial documents) and NewsML (the XML standard used by news and media publishers) are provided as examples of well accepted standards. To preserve the power of XML expressiveness these standards have all the properties mentioned above: variations in choices and number of occurrences of sub-structures, depth of hierarchy, mixing of meta-information and content, recursive tag naming, and semi-structured data.

It turns out that XML could be the ultimate 'information model' that computer scientists have been seeking since the inception of the industry. It has many features that other models don't have (with the possible exception of the object model, although it remain too constrained by its typing system). The model is self-describing. By looking at the document we see its structure and its content. The tag names carry the semantics and can be interpreted as such by applications and even human readers (provided tags are properly chosen). The XML data are so versatile that XML schemas have been invented which define the “grammar” of the documents to be “valid” in a particular domain. Since schemas are not mandatory on the one hand, and they do not in general reduce the complexity of the instances on the other, the immense challenge of effectively and efficiently querying documents remains.

How Do You Query XML Documents?

The answer is easy: use the XQuery language as defined by the W3C consortium. Which leads to the next question: how to execute such a query on a collection of millions of documents and still get answers in less than 1 or 2 seconds? The XQuery standard does not impose any limitations on the way XML structures are accessed and tested. Moreover in the extensions that the consortium is currently preparing there appears a new set of free text search operators which make the access even more complex (Xyleme implemented those operators in the first release of the product (in 2002) because they have been required by most of the use cases presented by our customers). These extensions are not surprising since XML, being a semi-structured model, implies a clear need for searching in any context of the XML structure, just like a full-text search engine would do, but at the whole document level.

So the question is now how to implement a fast execution of XQuery++? Some vendors propose using a classical relational database coupled with a classical full-text engine as the appropriate solution. But this approach is not viable for a general XML content system. The drawbacks are obvious:

- Complexity of managing two separate systems
- Separation of data (stored in database) and text (in search engine) is difficult and inflexible
- Performance is inherently poor, because of the overhead induced by the impedance mismatch with the XML model
- Classical indexing does not offer a solution, because the potential number of paths inside an XML structure is huge and it would not be realistic to create as many indexes as there are possible paths
- Functions are limited to what the data model can support. For instance only some queries would be allowed and the granularity of the result would not be supported beyond a certain level of structure

In conclusion, querying XML without losing its power needs a brand new approach in the database and indexing architecture.

Xyleme Server Architecture

An Index Combining Structure & Content

Xyleme introduces a new way to index XML documents. The server indexes in a single space both the full structure of the loaded documents and their content. The index is created and maintained (updated) in real time without any need of external directives.

The index knows for each tag its position in one or more documents. It knows also the hierarchical relationship between the tags. This information provides the ability of finding and filtering all paths which are used in an XQuery.

In addition to the structural information (tag positions), each PCDATA is also indexed. By default, a PCDATA is interpreted as a sequence of words. Each word is stemmed according to parameterized linguistic options and the stemmed root is recorded in the index. For each stemmed word the index knows to which documents it belongs and its exact position in the XML structure. By combining the path localization (sequence of tags) and the words reached by each path of the XQuery, the index has enough information to solve the predicates (including free text search operation) of the WHERE clause whatever their complexity.

When a document has been selected then the construction of its contribution to the result of the XQuery can be completed. Again the index gives the exact positioning of the nodes or PCDATA which are requested by the RETURN clause. From this positioning information how does one retrieve the actual information to build the result structure?

To achieve this extraction in an efficient way we need a direct access to any Node of the XML document. This is where the storage system plays an important role. Only a native XML storage system can meet this requirement.

A Native XML Storage System

Xyleme clusters documents within disk pages. It tries to store a full document in a single page (this is, in fact, a compacted binary representation of the original document). If the document cannot fit in a single page it is split in a balanced way into pieces, each piece being stored on a disk page. When the document is large (more than a given number of pages) a physical index records the page addresses of all its Nodes.

This organization ensures that a Node (whose localization has been computed by the index described in the previous section) is accessed at the cost of a maximum of 2 disk IOs. And having this clustering strategy coupled with the cache for rapid page access, most XQuery evaluations won't require more disk IOs.

A Native XML Storage System

How can Xyleme ensure a response time below 1 or 2 seconds whatever the complexity of an XQuery and the size of the queried collections?

We have seen how fast the computation of the results (the RETURN clause) is. What about the selection part of the XQuery (the FOR and WHERE clauses) which intensively use the index? The key factor which explains the performance is the fact that the index is kept in memory as opposed to being managed on disk with a cache. This design choice guarantees a selection phase in a few milliseconds whatever the paths which are visited and the words which are tested. We could imagine approaching a comparable performance with a query optimizer based on XML schema definition and a specific set of classical indexes. We have already talked about the first drawback of this solution which is to optimize only a subset of the possible queries. But a second major drawback is the constraint imposed by the schemas. The Xyleme query manager draws full benefits from the described architecture without using XML schemas.

For a model such as XML, we suggest it is a bad idea to base the optimization on the schema of the data. First of all because XML, by its nature, introduces too many variations of the actual data to allow you to derive an efficient structure based on the schema, and second, in the rapidly changing world of XML, fixing a schema entails frequent and very costly reorganizations of the database due to the schema evolutions. Finally there are many situations where the documents have no schemas or they are loaded despite the fact that they are not valid (this occurs frequently for applications which use multiple stages to transform and enrich documents which remain non-valid during a long period of time, but which must be still query-able).

Update Performance

Before being able to query a collection of documents they must be stored into the collection. Further, most documents are updated multiple times during their life cycle. Upgrading and updating indexes in a classical database is time consuming, and for real time systems which require documents to be query-able as soon as they are loaded this update time can become prohibitive. This problem is exacerbated further when you consider that in most classical database updates resources are locked during the update thus forcing queries to wait until the updates are committed (and XML documents can be quite large).

The in-memory index architecture proposed by Xyleme avoids these problems. The index is updated in memory in a very fine grained manner. The synchronization between queries and updates does not require long term locking. When a conflict occurs on a document the query sees the previous stable version of it without being locked.

With the index being maintained in memory how can the system recover after an un-planned abort? The index updates are recorded periodically in an incremental way to the disk. When a server restarts after a crash the last saved version of the index is reloaded, then it is resynchronized by retrieving from the store the documents which were committed between this last saved version of the index and the time of the crash. Those documents are re-indexed and the server is ready. This protocol guarantees no loss of information. Due to the speed of the store and the in-memory structure, this restart procedure is extremely fast.

With this infrastructure Xyleme can support a loading/update rate of about 1.5 GB per hour and per processor. As a matter of fact we will see in the next section that Xyleme Server can be distributed over several processors and in this case loading of documents can be done in parallel on these processors. With 2 processors for instance you can reach a loading rate of 3 GB per hour. This includes both storing the XML documents and indexing them in real time without blocking the queries which may run in parallel.

Scalability

With an in-memory index, the question naturally arises: "well, the in-memory index is an optimal approach for solving performance and flexibility challenges (as explained above), but what about memory consumption?" The index in memory is very compact. Even if it indexes both the full structure and the full content, the size (validated through extensive experience) varies from 40% to 70% of the size of the original documents. When an index is full the Xyleme infrastructure easily allows the addition of a new one on another process. The process can reside on the same processor or more likely on another processor. This incremental extension is transparent from the applications point of view.

By adding a new processor you also increase the performance of the system not only for loading and updating documents, as explained in the previous section, but also the performance of the queries. The reason is Xyleme will run a query in parallel over all active e processors. Each processor manages a partition of the data (and of the index) and the query manager computes an execution plan which is distributed among the interested partitions. In the Xyleme jargon we call such a partition a "repository".

To summarize, an instance of Xyleme manages as many repositories as necessary. A repository belongs to a process. The processes can be distributed over different processors. Each repository is able to load and query in parallel the partition it manages. A query execution can be distributed over the various processors. A repository can run several different queries on several threads. The system can thus be tailored within 3 dimensions to achieve the required performance: (1) the maximum size of an index managed by a repository (the memory size accessible by a process); (2) the number of processes and (3) the number of processors.

Xyleme provides all necessary tools to dynamically add a repository, to define the physical distribution of the logical collections (which are called “clusters” in the Xylem jargon), and the ability to reorganize this distribution. For instance to load and query 100 GB of documents you can decide to use 5 processors, each having 16 GB of memory. But without changing the applications you can at any time add processors to increase the overall throughput of the system. You can for instance split the size of an index in two 8 GB parts, each part being managed by a repository on a different processor. In this case the collection stored on this partitioned index will be loaded and queried more efficiently.

When the number of repositories becomes very large you may define a hierarchy among the repositories to limit the consumption of memory and processors, as explained in the next section.

On-line / Off-line Repositories

As the amount and type of XML within an organization grows it is natural that over time some of the content is only accessed occasionally and then typically for non-transactional purposes. In this case, the organization wants to have ready access to the data but their usage becomes less frequent and a latency to access them is acceptable.

The architecture of Xyleme not only allows you to dynamically add repositories but also allows you to take a repository off-line as well. These “archived” data are still accessible, but at the price to restart the repository which manages them. Restarting a repository is transparent for the applications and requires no intervention by the administrator or user. There is simply the delay associated with the re-loading of the index from disk (which takes from one to a couple of minutes depending on the size of the index).

With this capability an organization can over time, migrate non real-time documents to archive collections and then put these collections on repositories which eventually can be moved off-line. This approach allows for effective use of expensive resources particularly when managing the verbose (though powerful) nature of XML documents. For instance a processor which managed a now off-line repository should become available for running another repository. By doing so, the memory and processor resources can be reduced. But to achieve this protocol the same disks must be potentially accessible by different processes.

This ability is offered by a Network Attached Storage (NAS) for instance and Xyleme fully supports NAS. In this case the dialog between the repositories and the disks is done via the NFS-v3 protocol. Since Xyleme Server is CPU-bound the overhead of NFS is negligible (which has been confirmed by careful measurement of production systems in a large Xyleme customer).

This type of installation has other very interesting properties. The administration of the whole system is simplified; the very reliable RAID disks in addition to the transactional system of Xyleme Server insures a high level of availability; a hard crash of a processor can for instance be recovered with a new processor for the same repository.

Conclusion

Xyleme has implemented a very efficient and scalable system to support XML without any restrictions and with the maximum of flexibility.

The solutions to achieve these challenging requirements are innovative but have been proven by real cases in mission critical systems that our customers were able to build using our technology.

Structure of two XMLStandards: FPML (partial schema)

<?xml version="1.0" encoding="UTF-8" ?>	<s6:periodMultiplier/>	</s6:referenceEntity>
<DataGuide xmlns:s6 =	</s6:paymentFrequency>	<s6:referenceObligation>
"http://www.fpml.org/2004/FpML-4-1"	<s6:rollConvention/>	<s6:bond>
xmlns:s8 =	</s6:periodicPayment>	<s6:couponRate/>
"http://www.ml.com/swaps/gcd/aurora">	</s6:feeLeg>	<s6:description/>
<s8:Message>	<s6:generalTerms>	<s6:instrumentId>
<s6:FpML>	<s6:additionalTerm>	<instrumentIdScheme/>
<s6:header>	<additionalTermScheme/>	</s6:instrumentId>
<s6:creationTimestamp/>	</s6:additionalTerm>	<s6:instrumentId>
<s6:messageId>	<s6:buyerPartyReference>	<s6:maturity/>
<messageIdScheme/>	<href/>	</s6:bond>
</s6:messageId>	</s6:buyerPartyReference>	<s6:guarantorReference>
<s6:sendTo>	<s6:dateAdjustments>	<href/>
<partyIdScheme/>	<s6:businessCenters>	</s6:guarantorReference>
</s6:sendTo>	<s6:businessCenter>	<s6:primaryObligorReference>
<s6:sentBy>	<businessCenterScheme/>	<href/>
<partyIdScheme/>	</s6:businessCenter>	</s6:primaryObligorReference>
</s6:sentBy>	</s6:businessCenters>	<s6:referenceObligation>
</s6:header>	<s6:businessDayConvention/>	<s6:referencePrice/>
<s6:party>	</s6:dateAdjustments>	</s6:referenceInformation>
<id/>	<s6:effectiveDate>	<s6:scheduledTerminationDate>
<s6:partyId>	<s6:dateAdjustments>	<s6:adjustableDate>
<partyIdScheme/>	<s6:businessDayConvention/>	<s6:dateAdjustments>
</s6:partyId>	</s6:dateAdjustments>	<s6:businessCenters>
<s6:partyName/>	<s6:unadjustedDate/>	<s6:businessCenter>
</s6:partyName/>	</s6:effectiveDate>	<businessCenterScheme/>
<s6:trade>	<s6:referenceInformation>	</s6:businessCenter>
<s6:calculationAgent>	<s6:allGuarantees/>	</s6:businessCenters>
<s6:calculationAgentPartyReference>	<s6:referenceEntity>	<s6:businessDayConvention/>
<href/>	<id/>	</s6:dateAdjustments>
</s6:calculationAgentPartyReference>	<s6:entityId>	<s6:unadjustedDate/>
</s6:calculationAgent>	<entityIdScheme/>	</s6:adjustableDate>
<s6:calculationAgentBusinessCenter>	</s6:entityId>	<s6:scheduledTerminationDate>
<businessCenterScheme/>	<s6:entityName>	<s6:sellerPartyReference>
</s6:calculationAgentBusinessCenter>	<s6:entityNameScheme/>	<href/>
<s6:creditDefaultSwap>	</s6:entityName>	
<s6:feeLeg>		
<s6:periodicPayment>		

Structure of two XML Standards: NewsML (partial schema)

```

<?xml version="1.0"
encoding="UTF-8" ?>
<DataGuide xmlns:s13 =
"http://www.w3.org/1999/xhtml">
  <NewsML>
    <Catalog>
      <Href/>
    </Catalog>
    <Duid/>
    <NewsEnvelope>
      <DateAndTime/>
      <NewsProduct>
        <FormalName/>
      </NewsProduct>
      <NewsService>
        <FormalName/>
      </NewsService>
      <Priority>
        <FormalName/>
      </Priority>
    </NewsEnvelope>
    <NewsItem>
      <Duid/>
      <Identification>
        <DateLabel/>
        <NewsIdentifier>
          <DateId/>
          <NewsItemId/>
          <ProviderId/>
          <PublicIdentifier/>
          <RevisionId>
            <PreviousRevision/>
            <Update/>
          </RevisionId>
        </NewsIdentifier>
      </Identification>
      <NewsComponent>
        <AdministrativeMetadata>
          <FileName/>
          <Property>
            <FormalName/>
            <Value/>
          </Property>
          <Provider>
            <Party>
              <FormalName/>
            </Party>
          </Provider>
          <Source>
            <Party>
              <FormalName/>
            </Party>
          </Source>
        </AdministrativeMetadata>
        <Duid/>
        <EquivalentsList/>
      </NewsComponent>
      <ContentItem>
        <Characteristics>
          <Property>
            <FormalName/>
            <Value/>
          </Property>
        </Characteristics>
        <DataContent>
          <s13:html>
            <s13:body>
              <s13:p/>
            </s13:body>
            <s13:head>
              <s13:title/>
            </s13:head>
          </s13:html>
        </DataContent>
        <Duid/>
        <Format>
          <FormalName/>
        </Format>
        <MediaType>
          <FormalName/>
        </MediaType>
      </ContentItem>
      <DescriptiveMetadata>
        <Language>
          <FormalName/>
        </Language>
        <OfInterestTo>
          <FormalName/>
        </OfInterestTo>
        <TopicOccurrence>
          <Importance/>
          <Topic/>
        </TopicOccurrence>
      </DescriptiveMetadata>
      <Duid/>
      <EquivalentsList/>
      <Essential/>
      <NewsLines>
        <ByLine/>
        <CopyrightLine/>
        <CreditLine/>
        <DateLine/>
        <HeadLine/>
        <NewsLine>
          <NewsLineText/>
          <NewsLineType>
            <FormalName/>
          </NewsLineType>
        </NewsLine>
        <SlugLine/>
      </NewsLines>
    </NewsItem>
  </NewsML>
</DataGuide>

```

About Xyleme

Xyleme, Inc. is a leading provider of learning content management solutions that enable single-source publishing of learning content. Xyleme is 100% XML-based to simplify course development and significantly reduce the cost of supporting a blended learning strategy for instructor-led and online delivery. The Xyleme platform was developed around two key principals: modularity and reuse. This enables learning content to be effortlessly repurposed and customized to create assessments, curriculums and certifications that are personalized for the individual learner. Xyleme is SCORM 2004 certified and can enable existing SCORM 1.2 and AICC Learning Management Systems to take full advantage of this standard.

For more information about Xyleme, visit www.xyleme.com

Office Location

Xyleme, Inc.
2060 Broadway
Suite 250
Boulder, CO 80302
Tel: (303) 872-0233