Performance and scalability of XML query processing

Gaétan Hains\textsuperscript{1} \quad Muath Alrammal \textsuperscript{2} \quad Mohamed Zergaoui \textsuperscript{3}

April 2011
TR–LACL–2011–1

Laboratoire d’Algorithmique, Complexité et Logique (LACL)
Département d’Informatique
Université de Paris Est Créteil (UPEC), Faculté des Science et Technologie
61, Avenue du Général de Gaulle, 94010 Créteil cedex, France
Tel.: (33)(1) 45 17 16 47, Fax: (33)(1) 45 17 66 01

\textsuperscript{1}LACL-UPEC
\textsuperscript{2}LACL-UPEC and Innovimax SARL
\textsuperscript{3}Innovimax SARL
Performance and scalability of XML query processing

© G. HAINS, M. ALRAMMAL, M. ZERGAOUI April 2011.
Performance and scalability of XML query processing

Gaétan Hains, Muath Alrammal and Mohamed Zergaoui
LACL Université Paris-Est and Innovimax SARL, Paris France.

ABSTRACT

This chapter discusses basic issues about the performance of semi-structured query processing in very large datasets. It is based on recent algorithm-engineering work, on the state of the art in performance management for XML query processing and on theoretical studies about the complexity structure of the querying problem. Its main conclusions provide a concrete view on the interaction between terabyte scale XML data, query complexity and current or future computer architectures. To provide a concrete and synthetic view of this diverse body of knowledge, the presentation follows a fictional use case whose characters face query problems of varying complexity that are set in multiple contexts and analyzed in 2010 before being projected to 2020 and 2030.


INTRODUCTION

From an algorithms and performance point of view, the development of XML (W3C 2008) is a convergence of raw text data with completely structured relational data. In other words XML looks like text but should be processed like relational databases through searches, queries and transactions. As a result, operations based on XML data are extremely demanding of query processing performance, either because a user must have instantaneous results on very large data sets or because an advanced algorithm for data-mining or machine learning must repeatedly issue queries on its reference dataset. This chapter is an attempt to synthesize the practical consequences of the latest research on scalable query processing for XPath (W3C 1999), the XML querying language.

When it comes to predictable and efficient implementations of query processing on very large data, some “basic” problems like scalable join are far from closed even for relational data. Work on relational query optimization is still being published several decades after the definition of the problem (Al Hajj Hassan & Bamha 2010). And with XML data, the XPath query language produces problems that are
theoretically of high complexity (TenCate & Marx, 2009). It is therefore no surprise that the performance issues raised by XPath queries are many and require intricate analysis and experimentations.

When processing very large data sets one is either pushed towards solutions that read data in streaming mode (Viglas 2005) or parallel algorithms (Gibbons & Rytter 1992, Tiskin 1998) that take advantage of the very large internal memory and processing speed that multi-processing systems can provide. To illustrate this duality in a simple manner assume that a 1TB document must be processed with computing units equipped with 2GB of memory. For this purpose, stream processing can build internal structures (stacks and heaps) of up to 2GB as if filling then emptying memory 500 times. On the other hand parallel processing could use 500 processors simultaneously so as to store the 1TB data all at once in its massive but internally distributed memory. The streaming algorithm is limited by its capacity for building internal work tables but can process arbitrarily large datasets, while the parallel algorithm must fight against the relatively high cost of moving massive data around the physical parts of a huge internal memory.

There are multiple theoretical and practical issues to explore when building such algorithms and using them in practice. These problems have given rise to much research over the last decade and should lead to increased understanding together with increasing efficiency in the coming decades. But as of 2010 we do not feel that scalability of XPath queries is completely understood, nor that existing implementations exploit all of our hardware’s capacity. To inspire interest in this line of research and also provide an overview of this specialized and apparently narrow area, we try here to show how ongoing work can build streaming or parallel algorithms for XPath queries that will in the long run alleviate much of the intrinsic performance overhead of XML data. Instead of surveying an evolving and diverse set of research works, we present the problems from a quantitative perspective through an imaginary use case solve practical instances of the XPath query problem. We thus show, to the best of our available knowledge, what types of problems are routine, difficult or intractable today and with projected data and hardware in 10, and in 20 years.

As a self-contained introduction to XPath, we present here TenCate and Marx’s almost complete fragment of XPath 2.0. This language is called Core XPath 2.0 (TenCate & Marx, 2009). We abbreviate its name to CXP2 and give here its syntax and semantics for reference. The reason for using a fragment of XPath is that the authors have proved that the whole language has an undecidable query problem. It allows one to program counters that turn out to be expressive enough to make the query processor enter infinite loops whatever the algorithm. The restriction to CXP2 is therefore reasonable and still leaves a very rich query language whose query problem has a very high complexity, as researchers are finding out in many ways.

The syntax of CXP2 queries is given by the following grammar with root non-terminal PathExpr.

```
Axis ::= 'self' | 'child' | 'parent' | 'descendant'
      | 'ancestor' | 'following_sibling' | 'preceding_sibling'
NameTest ::= QName | ''
Step ::= Axis '::' NameTest
```
The semantic function \[ \[ Q \] \] takes for input the CXP2 query \( Q \) and is evaluated on an XML document \( D \) whose nodes are the elements of set \( \text{dom} \) and are numbered as if in their SAX ordering. The result is a set of pairs \( (x, y) \) such that the path from \( x \) to \( y \) satisfies \( Q \). This definition is more general than the usual assumption that \( x \) is necessarily the root of \( D \). But it is theoretically cleaner and respects the official XPath 2.0 definition. In summary, a query’s result is, for any initial node, a set of nodes in the document. The paths to those roots are described by \( Q \) and the set of nodes is thus a set of references to sub-documents. The basis of the semantics is the relation \( x(\text{Axis})y \) where \( \text{Axis} \) is either self, parent, child … the basic relations on nodes of \( D \) that XPath can express.

The treatment of variables \( \$i \) and loops in CXP2 requires an auxiliary parameter \( g \) which is an “environment” finite map of variables to nodes. If \( g \) is an environment and \( x \) a node then \( g[\$i := x] \) is the new environment where \( \$i \) goes to \( x \) and the rest is unchanged from \( g \). The value of a NodeRef expression is written \( [a]g,x \) and is \( g(a) \) if \( a \) is a variable or \( x \) if \( a \) is the dot (.) operator. The meaning \( [ \text{TestExpr} ]_g \) of a filter is a set of nodes: those for which one can find a solution for \( \text{TestExpr} \).

The meaning of CXP2 queries is given formally by the semantics below. Where we state that a pair of nodes satisfies \( x(\text{Axis})y \) we mean that they are located in the document according to the Axis relation: parent, descendant etc. For node \( y \) to satisfy NameTest=QName means that \( y \)’s label is QName. Any node \( y \) satisfies NameTest=\*.

\[
\begin{align*}
[| \text{Axis} :: \text{NameTest} |]^g & = \{ (x,y) : \text{dom}^2 \mid x(\text{Axis})y \text{ and } y \text{ satisfies NameTest} \} \\
[| \cdot |]^g & = [| \text{self} :: * |]^g \\
[| \$i |]^g & = \{ (x, y) : \text{dom}^2 \mid g(\$i) = y \} = \{ (x, g(\$i)) \mid x : \text{dom} \} \\
[| ( ) |]^g & = \emptyset \\
[| R / S |]^g & = [| R |]^g \circ [| S |]^g \text{ relational composition}
\end{align*}
\]
It is relatively easy to program a correct CXP2 query evaluator from the above equations, but this is no guarantee of a scalable or even efficient implementation. Minimizing temporary storage for stream processing, or minimizing non-local data access for parallel processing requires a serious analysis of the data structures involved. There has been much research on the design of query algorithms in the last decade and this work is continuing. The reader will find a description of ongoing work, relevant bibliographic references and proposed algorithmic solutions in (Gou & Chirkova 2007a, Gou & Chirkova 2007b, Alrammal, Hains & Zergaoui 2009) concerning stream processing for XPath queries. Similar information can be found in (Gottlob, Koch & Pichler 2005a, Gottlob, Koch & Pichler 2005b, Li 2006) concerning parallel processing applied to XPath queries, a less mature but closely related stream of research. The interested reader will also find a list of two dozen current publications on both topics in the additional reading section at the end of this chapter.

The next sections explain and quantify our knowledge about XPath processing performance in a lively way, through a broad use-case. It is the story of Ariane, Baptiste, Christine and Damien, co-workers and fictional users of XPath query processing engines. This use-case pure fiction but its technical content is based on real or highly plausible data, reasonable estimates for technology’s progress and the best of our knowledge on algorithms. The problem of query processing is multi-dimensional and very much data dependent but the values outlined here give a realistic order of magnitude for all the important factors affecting this key resource of the future: speed of access to massive data.

Technical terms not defined in the text are defined in the Key terms & Definitions section at the end of this chapter. It is possible to follow the general discussion on our use-case by skipping the Explanation sections below. Those sections provide details of our quantitative estimates that constitute the thread of the use-case.
**Figure 1**

Actors of project 1SARX

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ariane</td>
<td>Mobile engineer on 1SCO's project for the world cultural archive 1SARX.</td>
</tr>
<tr>
<td>Baptiste</td>
<td>Ariane's boss, EMEA division leader at 1SCO.</td>
</tr>
<tr>
<td>Christina</td>
<td>Baptiste's boss, 1SCO chief information officer.</td>
</tr>
<tr>
<td>Damien</td>
<td>Christine’s internal customer, 1SCO’s project leader for 1SARX.</td>
</tr>
</tbody>
</table>

**XPATH PROCESSING IN 2010, A QUANTITATIVE EXAMPLE**

**PROJECT 1SARX: XPATH FOR THE MASSES**

Our use case begins with Ariane. She is based in Paris and is “mobile IT engineer” at 1SCO (pronounce 1-S-Co). She must constantly travel her designated region of Europe - Middle-East - Africa or EMEA, to setup the institution’s newest service for citizens of the world. This service is a digital XML archive of the world’s cultural heritage and cultural production made available to the largest possible subset of the world population. This project and the archive itself have code name 1SARX (pronounce 1-S-Arch) and its core technologies are XML and XPath. In order to simplify access for users with light terminals, photo or video documents are not stored by 1SARX, yet they can be indexed in the archive. 1SCO develops this service for entry in service during 2010 and should maintain it at least until 2030.

**ARIANE’S USER: BANDWIDTH-BOUND**

Ariane spends most effort on areas where the smartphone is the most common terminal in use and users access 1SARX with internet access bandwidth that barely reaches 1Mb/s. Her goal is to estimate and maximize the efficiency with which her end users can search 1SARX with XPath queries issued from their smartphone.
Ariane restricts queries to the *forward XPath* language fragment. This allows streaming and avoids most heavy processing while serving many user needs like keyword searches, sub-document keywords, sequences of keywords etc. Ariane’s user queries the archive from his smartphone. If the technical solution was to use stream processing by the smartphone itself, then processing would be limited by the terminal’s tiny memory and most of all by the time needed to just stream through the archive. Indeed, it would take 88 days just to do so at current mobile phone download speeds (see explanation in the next section).

So queries must be supported by local servers that users access through their smartphone. Those servers must be kept up to date from a central copy maintained at 1SCO but we will assume this problem can be efficiently solved, which may or may not be true but seems plausible given that updates should be much less frequent and smaller than queries and their results. As we explained, the chosen technical solution is to have a national server store a copy of the archive for its citizens who connect to it through the mobile internet. That server can be an economical mainstream machine but it should be equipped with the fastest possible disk drive. The proposed user session is as follows:

1. The user sends a first query and the server answers within two hours with a summary of results.
2. The user refines his search with a second query which is answered this time within a few minutes.
3. Finally the user issues his final query whose result is 125MB large and is downloaded over mobile internet in 15 minutes.
4. The user can exploit this data locally on his smartphone.

Justifications for the above estimates are given below in the *Explanations for Ariane’s problem in 2010* subsection. For the next few years Ariane is satisfied with the 1SARX architecture for her region. It provides her users an affordable and attractive solution by answering random XPath queries within 3 hours of delay and no more than 3 times a few seconds of user attention. Moreover the architecture assumes only common hardware and networks, with the exception of a high-end disk drive on the server. Ariane can dedicate herself to the implementation and maintenance of this solution with the confidence that the 1SARX requirements can be met for EMEA during the next decade.

**EXPLANATIONS FOR ARIANE’S PROBLEM IN 2010**

We have set the size of the archive at 1TB because that is about ten times the current size of Wikipedia’s English content if provided as an uncompressed XML file: indeed on web page (Wikipedia 2010a) file `pages-articles.xml.bz2` is said to uncompress to 20 times its size of 6GB. At current mobile phone download speeds of about 1Mb/s, this explains the 88 days delay for streaming the archive that Ariane must avoid.

A typical user session is a sequence of 3 queries that are successively more selective and hence successively less costly in computation and communication. We estimate that the first query may have a
selectivity of up to 5% i.e. it produces an output that is $1/20^{th}$ the size of the XML archive or approximately 50GB. This is not true of pure keyword searches in general but with XPath operators like Axis descendant and NameTest *, even this estimate of 5% can be conservative.

If a local copy of the archive is stored in a local server then a first pass at the data must itself be a streaming query processing. This first pass could become a bottleneck unless the disk transfer rate is very high. We learn from (Wikipedia 2010b) that currently The fastest enterprise HDDs spin at 10,000 or 15,000 rpm, and can achieve a sustained transfer rate up to 1 Gbit/s. So we assume that the local servers are equipped with such high-end disk drives. This explains the delay of 2 hours for the initial query: all of the 1TB must leave the disk at 1Gb/s for stream processing on the server. The resulting output size is 50GB as estimated by the 5% selectivity. Now this output must go back to disk and a second query must stream through it again on the server. We may dismiss the few minutes that this takes. The result of the second query is 5% of 5% of the archive or 2.5GB which can hold in the server’s internal memory. As a result the third and last query can be processed in memory and is thus much faster, so that its duration is negligible for our estimate. The result of the third query is 5% of 5% of 5% of the archive or 125MB which the user must receive through the mobile internet to be stored on his smartphone. Indeed the smartphone can hold this size of data and, most importantly the 125MB ~ 1Gb can be transferred in about 1000 seconds at the mobile internet’s speed of 1Mb/s which explains the 15-minute delay for Ariane’s user.

In this part of the scenario we have assumed that a single user consumes 2 hours of compute time on the local server so that no more than 12 users could be served in a 24-hour day. But since that machine was assumed to be standard and hence relatively cheap, it is possible for most countries to buy many of them and spread the load over many servers. It is also possible to overlap many queries on a single disk stream (as if they were combined with the or operator of XPath) to take advantage of the fact that in-memory processing is much faster than disk access. So the actual architecture may look like a farm of very fast disks shared by inexpensive servers in sufficient quantity.

**BAPTISTE’S PROBLEM: FULL XPATH QUERIES, TOO COSTLY FOR NOW**

Baptiste is EMEA division leader at 1SCO. He is supervisor to Ariane and is also in charge of the more demanding customers who wish to access the archive with queries from a larger fragment than Ariane’s forward XPath. For example these customers are interested in queries involving the ancestor or following_sibling axes. This wish is understandable because full XPath is a published standard and should be processed as efficiently as possible. The technology for doing it is available and uses a parallel machine with 256 processing cores (64 quad-core processors) with a price tag in the many dozens of K€. Baptiste’s budget for the 1SCO/EMEA could afford one such system for internal use. But as a general service to citizens the parallel machine would be overwhelmed by simultaneous queries if, as hoped, the service became popular. One other option would be to use a cloud-computing service but this turns out to be just as costly for continuous 24hr/day service. One other option would be to buy many 256-core machines but that is beyond the 1SCO/EMEA budget. As a result Baptiste decides to drop this
service for the current decade and concentrates on supporting the basic stream-processing service that Ariane is responsible for. We will see later that he may change his mind in the future.

EXPLANATIONS FOR BAPTISTE’S PROBLEM IN 2010

The more general (non-forward) axes in XPath prevent the use of streaming algorithms. This is caused by the fact that random dependencies between document nodes involve non-linear access to the dynamic data structures. It follows that for all practical purposes the XML document being processed must be stored in memory. Currently, all research concerned with stream-processing of XPath concentrates on restrictive fragments like forward XPath as seen in (Alrammal, Hains & Zergaoui 2009) and in the bibliographic references thereof.

We have assumed that Baptiste’s target machine is a multiprocessor whose processors are quad-core CPU’s with a generous 4GB of RAM per core. A 16-processor or 64-core machine of this type could thus store 256 GB of XML document in memory for sequential random access or parallel processing of a general XPath query. But that is not large enough for the current archive size and complex hybrid solutions mixing streaming with parallel processing would seem to be necessary for this. On the other hand Baptiste can consider a 64-processor machine capable of holding the needed 1TB in memory. But as explained above this solution is too costly for a general customer service because of the machine’s price in the hundreds of thousands of €. Moreover, if only one system is available it will not suffice during peak hours and its cost precludes the purchase of enough physical copies of itself.

CHRISTINE’S PROBLEM: SIMULATING AND OPTIMIZING QUERIES ON THE ARCHIVE

Christine is Baptiste’s boss. She is 1SCO Chief Information Officer and is in charge of organizing the 1SARX global operations on a monthly basis. She collects performance meta-data for all user queries, the queries themselves (without their outputs) and analyzes this on a monthly basis. She uses a supercomputer to simulate a large number of user queries and so help 1SCO improve its query processing algorithms and architecture.

Christine receives the worldwide reports of queries on a monthly basis. This data consists mostly of the queries to the very popular smartphone service that Ariane and others have set up in all regions of the world. There are about 6 billion such queries per month and the monthly report is about 300GB large. She must test new versions of the current streaming processing algorithm used by Ariane’s servers. The results of these tests are immediately used to update the worldwide servers and improve performance, if only by a small percentage each time.

Christine is equipped with a medium-size supercomputer: 1000 quad-core processors with 16GB of RAM each and a high-speed interconnect and high-end hard disk drives like those on Ariane’s local servers. As a result Christine is tempted by the following straightforward setup: feed the XML archive data coming from the disks to the 4000 processing cores and let each one execute a copy of the (new) stream-processing algorithm. This would indeed provide a 4000-fold speedup over the daily operations but that is
not sufficient as processing time could climb up to 3000 hours even if only one user-query out of 1000 were used for optimization testing.

Christine’s technical solution is explained in the next section and amounts to side stepping the disk. It is indeed possible to load the whole archive into the supercomputer’s internal memory. Using this technique Christine can complete her monthly tests in at most 2 days. She is satisfied with this solution and will encourage her parallel processing specialists to improve it so as to reduce the delay even further.

**EXPLANATIONS FOR CHRISTINE’S PROBLEM IN 2010**

We assume that every one out of 100 citizens conducts a search per day on the smartphone service. The estimate of one query per day is pure speculation but given the several minutes of user time + 2 hours of waiting time it is unlikely that active users are willing to issue more than one query per day. This amounts to 1.8 billion searches per month if we assume a population of 6 billion. Moreover, per our scenario the searches are triples of queries and this amounts to 5.4 billion individual queries per month. The estimate of 1 user per 100 citizens is very conservative as it would mean that approximately the population of France is an active while the rest of the world is not. In other words we assume that one citizen out of 100 issues a triple query every day of the month hence about as many queries per month as there are citizens in the world.

The report Christine receives includes for each query: a unique query identifier, country of origin, date + reception time, processing time without transmission, output size, the query syntax. This data should constitute about a dozen 32-bit words or approximately 50B, hence the size of the monthly report received by Christine: 50 B/query-record * 6 Gquery-records / month = 300 GB/month.

Christine’s monthly tuning operations require that a large subset of the user queries be re-processed with variants of the streaming algorithms that are installed on 1SCO’s servers worldwide. The queries are thus processed again in streaming mode, despite the availability of a much larger computer at 1SCO headquarters. Instead of devoting this computer to the irrelevant task of speeding up single queries (irrelevant because this machine may not be dedicated to individual user queries), it is used to calibrate and improve the stream processing algorithms that are actually put in service. Its multiple disks and processing units are used to re-process more than one query at a time.

Let us assume the new algorithms are a few percent faster, or sometimes slower than those of the previous month but then one re-processing test requires approximately two hours of (mostly disk data rate) delay, as in Ariane’s architecture. On a supercomputer with 4000 computing cores, e.g. 1000 quad-core processors, each cores can process a separate streaming algorithm on the single data stream coming from the fast hard disk. Each processing task would last two hours as predicted previously and there would be 4000 completed every two hours or 2000 queries/hour. Even if Christine processed only 1 out of 1000 user query to test her algorithms, i.e. 6M queries, that would require 3000 hours or 125 days, much longer than a month. That method is therefore impractical.
But there is a straightforward optimization that Christine may use. On a distributed-memory supercomputer the amount of internal memory is very large, proportional to the number of processors. In the above estimated system there could easily be 16TB of RAM available, enough to store the complete archive XML document. Given that memory is shared but individual tests are performed in parallel, this would involve some careful programming or tuning but appears quite feasible: read the document in sequential fashion from the local RAM bank closest to the program pointer following that stream. Then distribute that data stream in hierarchical fashion to every core in the machine. This distribution should be faster by at least a factor of 70 and here is why. By using the supercomputer’s internal network this transfer could proceed at about 9GB/s: The Cray XT supercomputers (Cray 2010) technical data announces interconnect on the XT6 and XT5 models: 6 switch ports per Cray SeaStar2+ chip, 9.6 GB/sec each. The transfer rate of 9GB/s on the supercomputer’s internal network is 9*8 faster than the disk-drives transfer rate of 1Gb/s. Using this technique it appears possible to improve the 900 hours delay at the very least by a factor of 70, without any non-trivial use of parallel processing. We therefore conclude that the improved technique incurs 3000 hours/70 or approximately 2 days of delay for Christine’s monthly batch of tests on 1 out of 1000 individual user query.

**DAMIEN’S PROBLEM: MORE COMPLEX PROCESSING**

Damien is project leader for 1SARX and reports directly to 1SCO top management. Having convinced them of the archive’s critical importance he is keen to manage wisely and develop citizen satisfaction and cultural value from their precious data set. With this in mind Damien uses time on a grid of large supercomputers provided by the world’s public sector and he runs machine learning routines on 1SARX to test ideas about the archive and extract information that is not explicitly visible through queries. In line with the institutions’ global objectives, Damien recruits top researchers and assists them in solving deep questions formulated as a machine-learning problem on the archive.

Each machine on the grid is of TOP-500 supercomputing size with 100 000 processing cores. But the supercomputers are geographically distributed and shared with other large users. So Damien’s main worry is to find available processing power on this grid, and move his archive to the necessary system for running his heavy computational jobs. As explained below the transfer does not incur more than a few hours of delay, which is reasonable for him given that grid users run batch jobs over weeks and that time slots may be known many days in advance.

Damien’s favourite algorithm is a genetic programming procedure for machine learning and is inspired by work of Ch. Vrain’s team (Braud & Vrain 1999). We explain the algorithm by an example general enough to be short but sufficient to outline the computation it generates. Assume a linguist is searching the archive not for explicit information but about correlations between unrelated parts of the archive. For example the linguist may search for all mentions of a certain Arabic philosopher, in every language covered by the archive. Even that basic question is more complex than an XPath query unless the linguist knows in advance all spellings of the philosopher’s name in every alphabet, which is unlikely. The linguist is able to generate a large list of small XML documents that do represent the philosopher’s name, and also a similar list for counter-examples like other philosopher names or similar-sounding Arabic
names etc. The machine learning algorithm then creates, almost at random, a “population” of XPath queries and computes a fitness function on each “individual” of the population: it applies each query to the XML archive and compares the result with the target lists of examples and counter-examples. The goal of the algorithm is then to maximize this fit between a query and its ability to cover the examples while avoiding the counter-examples. The algorithm repeatedly proceeds by ranking queries with the fitness function, eliminating the worst ones, combining the best ones and thus creating a new “generation” of queries. If fitness raises above a certain threshold the result is returned to the linguist who may then use the resulting (usually complex) query to “explain” the concept he is studying.

Damien wants to build an architecture to support this kind of learning algorithm. By the algorithm’s description it is clear that the key parameters for performance are the number of individuals in the population, the number of iterations (called “generations” in genetic programming) and the time required for one iteration. Damien has no control over the first two parameters but he will use his best algorithms and computers to minimize the third one. The analysis of next section estimates that the genetic algorithm may evolve query-individuals at the rate of 100 generations per hour or 2400 generations per day for populations of 1000 individuals. This is quite satisfactory and assuming the algorithm converges, Damien could return a few days later with a complex XPath query that closely matches the researcher’s data who would study that result with attention. The financial cost of this type of processing is currently prohibitive but its feasibility is enough to convince Damien that the archive’s long term value goes much further than an XPath query service.

EXPLANATIONS FOR DAMIEN’S PROBLEM IN 2010

The first problem Damien must solve is that each time a supercomputer is available he must transfer the 1TB archive to that system. If the grid’s geographical sites are connected with a conservative 1Gb/s link then Damien is able to complete the transfer in about 8000s or approximately 2 hours which is very satisfactory and avoids the need to store multiple copies of the archive at multiple sites where, over time, confidentiality may become a problem.

The genetic machine learning algorithm itself is beyond the scope of this paper and its performance is very much dependent on the application domain. For the sake of our estimates we will assume that the generations are filled with 1000-query populations. As seen in Christine’s problem, the full archive can be loaded in the internal memory of a medium-sized supercomputer and on the large ones with 100 000 cores there should be up to 400TB of RAM available.

Damien will then load about 400 copies of the archive in memory with about 100 000 cores / 400 copies = 250 computing cores available to work on each copy. It is not possible to assume that the 250 cores will necessarily speed-up an individual query by a factor of 250 because current understanding (Gottlob, Koch & Pichler 2005a) is that the parallel speedup factor depends not so much on the document’s shape as on query factors such as the depth of nesting on transitive axes (ancestor composed with descendant composed with sibling …). We will thus be conservative and assume that the 250 cores acting in parallel can only provide a speedup factor of 25 when processing a single query, compared to its stream-
processing by one core. The precise manner of obtaining this with a parallel algorithm is the subject of much ongoing research. In Ariane’s context, one core completed a stream processing in two hours because of the disk’s speed. But taking data from RAM memory should be about 100 times faster. Hence we estimate that the 2-hour delay can be accelerated with faster access and parallel processing by a factor of $100 \times 25 = 2500$ down to about 3s per query or 1/3 of a query per second. Recall also that there are 400 such blocks of 250 cores with one RAM copy of the archive, all proceeding in parallel. So in total the supercomputer should process about 400 blocks * 1/3 query/s or more than 100 queries per second.

In terms of the genetic algorithm this means that a generation can be evaluated in a few dozens of seconds. We avoid describing here the necessary sorting algorithm for ranking query fitness but that can be completed in much less time since it amounts to sorting a few thousand numbers. A slightly more costly step is the “cross-over” combination of selected queries but fast internal communication devices ensure that the supercomputer can complete this step in a matter of micro-seconds. As a result we estimate that the genetic algorithm can evolve at the rate of more than a hundred generations per hour. This justifies our estimate of the speed of Damien’s algorithm.

**XPath Processing in 2020 and 2030, Where Will We Be Then?**

Our use-case of XPath processing problems has been analyzed with data dated 2010 like this chapter. Some of its conclusions are rather positive if imperfect:

1. Ariane’s users can have an affordable and reliable stream-processing service for forward XPath queries if they are willing to wait two hours, refine their query twice and then download the result in 15 minutes.
2. Christine can conduct extensive testing of new query-processing algorithms every month, using a large subset of real data and within 2 days. This requires a 4000-core supercomputer whose cost could be not more than 1M€.
3. Damien is able to run a machine-learning algorithm that inverts the problem: given sets of wanted responses and unwanted counter-examples, construct an XPath query which fits those constraints as well as respecting all related information found in the XML archive. This can be completed in a few days on a very large supercomputer which may be hosted elsewhere: the data transfer time is negligible for this purpose.

Our main negative conclusion has been with Baptiste’s objective of providing a service for full XPath (e.g. CXP2) queries. The demand for parallel processing makes this uneconomical, at least for deployment on a grand scale.

If we project ourselves over the lifetime of project ISARX, we find that many key figures related to our estimates will change. For example the size of the archive itself, the number of users and the speed and capacity of hardware components. In this section we reflect on what this evolution should mean for Ariane, Baptiste, Christine and Damien. The conclusions we reach below are no deeper than the above analysis, so that the actual evolution of parallel algorithms may add one (positive) dimension to all of our conclusions. But this chapter remains to be written.
So we will now enquire whether the feasible problems of today going to remain so, and whether Baptiste’s full XPath query problem will become economically?

One factor that influences all of our use case is the size of the archive. We will estimate that the ISCO archive will grow at the same rate as the so-called total digital content of the world. According to Wikipedia under “Exabyte” we find that

1. As of May 2009, the size of the World’s total Digital content has been roughly estimated to be 500 billion gigabytes, or 500 exabytes.
2. Earlier Berkeley studies estimated that by the end of 1999, the sum of human-produced information (including all audio, video recordings and text/books) was about 12 exabytes of data.

This growth rate of 12 to 500 EB in ten years amounts to a growth factor of 45% per year. At this growth rate the 1SARX dataset will reach 40TB in 2020 and approximately 2PB in 2030.

Another general factor is the world population, potential users of the 1SARX service. In 2010 we have conservatively put this number to 6G. Under “World Population” in Wikipedia we find numbers going back to 1804 whose trend is extrapolated to reach 8G in 2025 and 9G around 2045. From this we roughly interpolate the population to 7.5G in 2020 and 8.5G in 2030.

ARIANE’S FUTURE: STREAM PROCESSING IN 2020 AND IN 2030
The main factors affecting Ariane’s technical solution have been: the speed of mobile internet and the delay to load the archive from disk to the server. The historical growth of home internet transfer rates has been relatively low. In 1995 we were offered 512Kb/s download rates and currently use home services that provide no more than a few dozens of Mb/s real download rates. These measurements may vary but they give an order of magnitude for the growth rate. Ariane’s users are limited by speculative current 1Mb/s rate on mobile phones. If the factor of approximately 20 observed between 1995 and 2010 for home internet is a guide, we can expect Ariane’s users rate to climb to a little more than 10Mb/s in 2020 and then 100Mb/s in 2030. The 40TB of the archive will no more fit into internal memory of a small server in 2020 than the 1TB does now. Nor will they be able to travel down the 10Mb/s link in 2020 or in 2030. As a result, Ariane’s user protocol should not change. Its limiting factor will remain the time to stream the archive from a high-end disk drive through the server for stream-processing. To estimate the data transfer rate of external 40TB storage in future years appears delicate because the exact technology is not yet clear for a high-end device in 2020 (rotating hard disks or solid-state devices), at least it is unclear for non-specialists like the authors of this chapter. Nor have we heard specialists make such long-term predictions. We will therefore rely on a simple application of Moore’s law: the current estimated 1Gb/s data-transfer rate will double every 18 months. As a result it will climb to more than 64Gb/s in 2020 and then 4Tb/s in 2030.

Recalling Ariane’s scenario, we therefore estimate that the main delay in 2020 (resp. 2030) for her processing task will be to transfer a 40TB (resp. 2PB) archive into a server at 64Gb/s (resp. 4Tb/s). This will then require approximately 80 minutes (resp. 60 minutes) instead of the current 2 hours.

BAPTISTE’S FUTURE: FULL XPATH PROCESSING IN 2020 AND IN 2030
The failure of Baptiste’s plan in 2010 is due to his inability to procure a parallel machine large enough to hold the archive in its memory yet cheap enough to be put in production in many instances of the same system. The need for an internal-memory (non-streaming) algorithm for full XPath queries hold true forever and will thus remain in 2020 (resp. 2030). If our current generously estimated 4GB RAM/core grows with Moore’s law it should become $4 \times 64 = 256$ GB/core in 2020 (resp. $4 \times 4000 = 16$ TB/core in 2030). So it appears that Baptiste will be able to purchase in 2020 a 160-core machine with 40TB of RAM enough to hold the archive in memory. In 2030 the 2PB of the archive could be stored in a 128-core machine if each core is provided with 16TB as we extrapolated. By those dates such sets of cores could be combined into a single microprocessor.

According to those rough and naïve estimates, it seems that parallel processing of the whole archive could become a reality if memory and processor technology continues to miniaturize and if the archive itself grows no faster than the world’s digital content. By 2020-2030 parallel XML algorithms and parallel programming tools should have matured enough and be well understood by many developers. As a result we conclude likely that Baptiste’s plan can materialize during the next decades.

CHRISTINE’S FUTURE: EXTENSIVE MONTHLY TESTING IN 2020 AND IN 2030

The main factors affecting Christine’s monthly batch computational problem are: the world population which is related to the number of monthly queries on ISARX, and standard machine parameters on a supercomputer. As the population’s growth rate is more modest than that of the machine parameters, it appears very likely that our estimates for Christine’s jobs will remain true and even improve in 2020 and 2030.

DAMIEN’S FUTURE: MACHINE LEARNING OF QUERIES IN 2020 AND IN 2030

Damien’s algorithm is the most speculative of all tasks considered here, yet possibly the most important. In the performance estimation about his machine-learning algorithms we have kept algorithm parameters separate from machine parameters. As long as a parallel algorithm can accelerate an XPath query by less than $1/10^{\text{th}}$ its allocated number of processing cores, the estimate will hold. Moreover the parameters of the algorithm and of the parallel supercomputer should remain independent and grow at their respective independent rates. The former are completely dependent on machine-learning techniques and their estimate is beyond the scope of this chapter. If the algorithmic parameters grow no faster than roughly Moore’s law then the estimates for Damien’s computation will not only remain but improve in the future. In a sense this means that Damien can consider his HPC architecture as a black box, or almost.

CONCLUSION: DATA SCALABILITY, PROBLEM COMPLEXITY, ALGORITHMS AND HARDWARE

The analysis presented in this chapter relates to many concerns of data-mining work. It addresses performance issues, yet it is strictly application-oriented in the sense of analyzing not only hardware, middleware or low-level software aspects but the algorithms that power any large-scale XML data-mining
activity. Luckily algorithm analysis and engineering has much progressed since the last two decades and we are now ready to build systems that are predictably scalable and yet provide rich high-level functionality on XML data.

The interaction of stream processing and parallel processing for XPath queries opens many possibilities which we hope to have clarified with our quantitative examples in this chapter. Ongoing and future research should improve the existing scalability and predictability of streaming algorithms. It should also refine the early parallel algorithms that already exist and demonstrate their feasibility on very large XML data sets.

The interaction between language fragments and performance deserves even more study than it has received until now. Once these questions have been completely clarified, XPath processing will become a fully reliable and scalable tool for more elaborate algorithms, as the naïve but practical case of Damien’s algorithm shows. The interaction of XPath processing with machine parameters appears to follow an encouraging trend, and algorithm engineering should be able to develop many useful data structures and libraries to exploit this potential scalability.

Open problems which we have not touched upon here are: constraint solving on XPath, XPath queries on incomplete information, quantum algorithms for architectures that may appear beyond 2030, adaptation of XML and XPath querying to non-textual data. We hope the ideas and calculations presented here will stimulate discussion and research on all of those questions.

The authors thank the editors and reviewers for their assistance with the publication of this chapter.

REFERENCES


**KEY TERMS & DEFINITIONS**

**XML document**: A document written in the extensible markup language. Its hierarchical structure is explicitly written by markup tags and the rest of the document describes pure content.

**DOM**: The domain object model of XML documents is a programming interface for navigating in the document which is assumed to be a tree structure stored in computer memory. This is corresponds to the classical view of sequential programming and is a reasonable approximation of parallel programming for XML documents.

**SAX**: The Simple API for XML is a programming interface for navigating the document in text-sequential mode i.e. in the *left-to-right depth-first search* (DFS) order of its tree structure. The advantage
of SAX over DOM is that it only requires working memory to store relevant parts of the document being processed. This corresponds to the situation of a stream-processing algorithm. The disadvantage of SAX is that of any external-memory algorithm, namely the lack of a random access to all parts of data.

**DFS:** Depth-first search traversal is an algorithm for traversing a tree by first visiting the root, then recursively traversing the leftmost subtree, the second leftmost subtree, etc. The sequence of lines in an XML document represent a left-to-right DFS traversal of the document’s DOM- or tree structure, if we account for the fact that the root is visited in the beginning (opening tag) and also at the end (closing tag).

**XPath query:** The XML Path Language is a query language for selecting nodes and parts of an XML document by specifying navigation paths down the tree structure of the document. The current version XPath 2.0 includes fragments of high computational complexity.

**Stream processing:** A form of algorithm where the full document only moves between permanent storage units, or is created/processed on the fly. Its advantage is reduced memory consumption and the possibility of treating documents of almost unlimited size. Its disadvantage is software complexity and restrictions on the applicable XPath fragment.

**XPath fragment:** In Logic and Computer Science a language fragment denotes a coherent subset of a given language’s expressions. When the whole language’s expressivity or computational complexity is found to be too high, computer scientists define language fragments to balance those two conflicting goals. With XPath the need for high-performance query processing or, almost equivalently the need to process queries on very-large documents leads to the definition such language restrictions. For example Forward XPath denotes a fragment where “navigation” may only go down the tree structure.

**Parallel processing:** Computing with multiple processing units acting simultaneously on the same data to reduce processing time. Due to synchronization and communication costs, there may be an optimal number of processing units beyond which parallel processing actually becomes slower.

**Scalability:** Quality of an algorithm whose time and memory space requirements grow moderately as a function on input data size. In practice a scalable algorithm requires space and time that depend only linearly (or not much more than quadratically) on the size of input data. This is explained by the theoretical notion of low-degree polynomial computational complexity.

---

**ADDITIONAL READING**


