Análisis y Alternativas para la Compresión de XML

Matías Bordese
Directora: Dra. Laura Alonso i Alemany

Agosto de 2009

Facultad de Matemática, Astronomía y Física

Universidad Nacional de Córdoba

Clasificación:

E.4 Coding and Information Theory: Data Compaction and Compression

Palabras clave:

sistemas distribuidos, xml, compresión, dataset, consultas

UNIVERSIDAD NACIONAL DE CÓRDOBA

Facultad de Matemática, Astronomía y Física

Licenciatura en Ciencias de la Computación

Análisis y Alternativas para la Compresión de XML

por
Matías Bordese

Resumen

El presente trabajo describe las distinas técnicas y herramientas disponibles para la compresión de documentos XML.

El uso de XML continúa creciendo, y se hace necesario explorar técnicas de compresión efectivas y específicas para documentos XML con el objetivo de reducir el uso de recursos (el ancho de banda requerido para el intercambio de datos, el espacio de disco requerido para almacenamiento, o el uso de memoria al momento de procesar y consultar documentos XML) sin comprometer la velocidad y flexibilidad.

El objetivo de este trabajo es el de investigar las razones por las que se hace necesario contar con métodos de compresión específicos para XML e identificar en ellos las técnicas que los distinguen y los hacen útiles para diferentes casos. También se presentan implementaciones existentes, analizando cómo cubren éstas las distintas necesidades y evaluando su performance.

Agradecimientos

A la Dra. Laura Alonso i Alemany, por la dirección del trabajo.
A la Nati Bidart, por su paciencia, sus revisiones y consejos.
A mi familia, que siempre me acompaña.
Y a todos mis amigos sacuyistas.

Índice general

Capítulo 1

Introducción

1.1. XML

El *eXtensible Markup Language*[1] (XML) se está convirtiendo en un formato estándar para el almacenamiento e intercambio de datos. Está basado en el *Standard Generalized Markup Language*[2] (SGML), uno de los primeros lenguajes de markup comerciales. Otro lenguaje de markup anterior es el *HyperText Markup Language*[3] (HTML), también subconjunto de SGML.

Todos estos lenguajes usan etiquetas (o *tags*) para delimitar los contenidos y la metadata. Es decir, que un documento puede contener tanto datos como información acerca del rol que ellos cumplen (*metadata*). Esta información adicional ayuda a las diferentes aplicaciones a procesar o presentar dichos datos. Pero, por qué surgió la necesidad de un nuevo lenguaje de markup? La realidad es que SGML y HTML tienen algunas desventajas. SGML es demasiado complejo de manejar. Y si bien HTML es la opción para la presentación de páginas web, tiene una extensibilidad limitada, además de que los tags predefinidos se usan más para describir la apariencia de los datos que su estructura. Sin embargo, para el intercambio de información en internet era necesario contar con un lenguaje de markup extensible, flexible y conciso. En función de esta necesidad, el *World Wide Web Consortium* (W3C) diseñó XML con la intención de cubrir los siguientes objetivos:

- XML debía ser utilizable en internet de manera directa.
- XML debía soportar una amplia variedad de aplicaciones.
- XML debía ser compatible con SGML.
- Debía ser fácil desarrollar programas que procesaran documentos XML.
- El número de características opcionales en XML debía ser mínimo, idealmente cero.
- Los documentos XML debían ser legibles por humanos y razonablemente claros.

[1] `http://www.w3.org/XML`
[2] `http://www.w3.org/MarkUp/SGML/`
[3] `http://www.w3.org/html`

- El diseño de XML debía prepararse en un corto plazo.
- El diseño de XML debía ser formal y conciso.
- Debía ser fácil crear documentos XML.
- La brevedad en el markup de XML era de menor prioridad.

Las dos virtudes más importantes de XML son el hecho de ser auto descriptivo y su extensibilidad. En XML los tags que delimitan elementos también pueden ser usados para describir los datos dentro de los mismos. Eligiendo de manera cuidadosa los nombres de los tags, un documento puede indicar la semántica de los datos, y en estos casos bastaría con compartir un vocabulario XML (o *tag set*) para intercambiar información.

La extensibilidad es otro de los beneficios de XML, reflejada en el hecho de que su sintaxis no define los tags, sino que un tag set puede ser extendido por cualquiera para su propio propósito, combinando tags para definir estructuras complejas de información. A diferencia de HTML, XML separa la semántica de la presentación. Los documentos describen cómo ser interpretados, dejando al consumidor el problema de la presentación.

Debido a su simplicidad, tanto en los conceptos como en la teoría por detrás, XML es utilizado para resolver numerosos problemas tales como el de proveer una representación neutral para datos intercambiados entre arquitecturas completamente diferentes, servir de intermediario entre distintos sistemas de software con un mínimo esfuerzo, o almacenar grandes volúmenes de datos semi estructurados.

Tal es así que en los últimos años XML se ha convertido en el estándar de facto para comunicar sistemas heterogéneos entre sí, y es utilizado para estandarizar el intercambio de información en varios campos, desde reportes de negocio hasta resultados de deportes, sin dejar de mencionar información de salud o educación.

Sin embargo estas ventajas que se mencionan no son absolutas. Uno de los inconvenientes de trabajar con documentos XML es su verbosidad. Y este incremento de tamaño afecta la cantidad de información que debe ser transmitida, procesada, almacenada y/o consultada.

1.2. SOA y XML

Con el surgimiento de XML evolucionaron otras tecnologías. Tal es el caso de *Service Oriented Architecture*[4] (SOA), un marco de trabajo que provee métodos para el desarrollo e integración de sistemas como servicios interoperables. Una infraestructura SOA permite a diferentes aplicaciones intercambiar datos con otras. La idea es separar funciones en distintas unidades o servicios, poniéndolos disponibles en una red de tal manera que los usuarios puedan combinarlos y reusarlos en el desarrollo de nuevas aplicaciones. Estos servicios se comunican a través de pasaje de datos de uno a otro, o coordinando actividades entre 2 o más servicios.

[4] `http://en.wikipedia.org/wiki/Service-oriented_architecture`

Un caso particular de SOA, pero tal vez el más representativo, es el de los *Web Services*[5]. Los Web Services pueden implementar una arquitectura SOA, haciendo accesibles bloques funcionales a través de los protocolos estándar de internet e independientemente de las plataformas y lenguajes de programación. Estos servicios pueden ser tanto aplicaciones nuevas como intermediarios (o *wrappers*) alrededor de sistemas ya existentes para habilitar su uso en una red.

Las especificaciones de un servicio web incluyen: un protocolo de comunicación, un lenguaje de descripción de servicios y un protocolo de publicación y descubrimiento de metadata. Normalmente se utilizan tecnologías basadas en XML para cada uno de los anteriores: *Service Oriented Architecture Protocol* (SOAP), *Web Services Description Language* (WSDL) y *Universal Description Discovery and Integration* (UDDI), respectivamente, lo que convierte a XML en la tecnología básica para la interacción entre servicios.

Por supuesto que el uso de XML brinda sus ventajas y desventajas: se consigue una total independencia de la plataforma y lenguaje, permite interoperabilidad y reusabilidad, pero los archivos XML son pesados y lentos de parsear. Y en estos casos el intercambio de documentos XML es bastante alto. La compresión es una alternativa para enfrentar este problema. Al mismo tiempo, también están apareciendo tecnologías para el parseo de XML y formatos binarios compatibles con XML para mejorar la performance.

1.3. Necesidad de compresión

El uso de tags significativos permiten que un documento XML sea auto descriptivo, pero también incrementan el tamaño del documento. Un dato en general viene rodeado de un tag de inicio y un tag de final, como se puede ver para "John" en `<FirstName>John</FirstName>` (línea 5) en el ejemplo 1.1. En algunos casos, los tags requieren más espacio que el propio dato.

Consideremos el siguiente documento de ejemplo:

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <Invoice ID="ST87301" date="2009-03-17" paid="true" >
3     <Customer ID="2576">
4        <Name>
5           <FirstName>John</FirstName>
6           <LastName>Smith</LastName>
7        </Name>
8        <BillingAddress>
9           <Street>200 University Ave.</Street>
10          <City>Waterloo</City>
11          <Province>ON</Province>
12       </BillingAddress>
13    </Customer>
14    <ProductList>
15       <Product ID="HI0741" >
16          <name>MP3 Player</name>
17          <Price>139.99</Price>
18          <Units>2</Units>
```

[5] `http://en.wikipedia.org/wiki/Web_Services`

```
19        </Product>
20        <Product ID="ZP6015" >
21          <name>CD Writer</name>
22          <Price>79.99</Price>
23          <Units>4</Units>
24        </Product>
25      </ProductList>
26  </Invoice>
```

Listing 1.1: Ejemplo de documento XML

Este documento contiene 19 ítems de datos y tiene 676 bytes. Si se almacenan los datos propiamente dichos en un archivo de texto plano como se muestra en el ejemplo 1.2, éste tiene sólo 124 bytes. Este valor incluso se puede reducir si usa un formato binario.

```
ST87301 2009-03-17 true 2576
John Smith
200 University Ave. Waterloo ON
HI0741
MP3 Player
139.99 2
ZP6015
CD Writer
79.99 4
```

Listing 1.2: Datos en texto plano

La repetición de tags hace que la versión en XML sea casi 5 veces más grande que la versión en texto plano. Aunque los archivos individuales que se transmiten sobre una red pueden no ser muy grandes, en general el número de archivos a transmitir suele ser tal que se consume un gran ancho de banda. En las circunstancias en que el ancho de banda es limitado y hay mucho tráfico, la transmisión de documentos XML puede convertirse en un cuello de botella para las aplicaciones. La preocupación persiste también cuando se usa XML para almacenar información, ya que la performance de las consultas está influenciada por el tamaño del archivo XML.

Surge entonces la necesidad de compresión de documentos XML para su eficiente almacenamiento y transmisión. Una primera alternativa es usar herramientas de compresión de texto general, como zip[6] o gzip[7]. Y si bien estas herramientas pueden reducir el tamaño de un archivo XML, no pueden explotar las propiedades inherentes a XML para obtener mejores resultados.

Existen varias estrategias para mejorar los niveles de compresión de documentos XML, aprovechando la estructura de árbol de los datos o agrupando los valores de acuerdo a la semántica del documento, por ejemplo. Pero para eso es necesario contar con un compresor que entienda la sintaxis de XML, y es por esta razón que surgen los *compresores específicos de XML* (o XML-conscious compressors).

[6] http://en.wikipedia.org/wiki/ZIP_(file_format)
[7] http://www.gzip.org/

Capítulo 2

Clasificación de Compresores

Al momento de elegir una herramienta para comprimir documentos XML podemos distinguir ciertas características generales que nos permiten clasificar las alternativas en diferentes grupos [Sak08].

Una primera gran división es la que separa a los compresores que hacen uso (es decir, son *"conscientes"*) de la estructura de los documentos XML (*XML-conscious compressors*) y aquellos que no. Este capítulo se centra en los compresores específicos de XML, entre los que se distinguen subclasificaciones según los siguientes criterios: dependencia de esquema, capacidad de soportar consultas, y si operan en línea o no.

2.1. Clasificación y características

Como se comentó, la clasificación más amplia que se puede hacer es en dos grandes grupos:

- **Compresores de texto general**

  Siendo los documentos XML archivos de texto, una alternativa lógica es usar herramientas de compresión general. Éstas tienen la ventaja de estar ampliamente disponibles y de ser independientes del tipo de archivo a comprimir. Sin embargo, la compresión de los elementos (o atributos) de un documento XML puede verse limitada debido a los rangos distantes entre las dependencias de elementos (o atributos) dentro del documento, es decir por el hecho de que la duplicación en que se basan este tipo de compresores no es necesariamente local para el caso de los archivos XML. Esta limitación se debe a que no hacen uso de la estructura del documento.

- **Compresores específicos de XML**

  En este caso se trata de compresores diseñados para aprovechar la estructura de los documentos XML y así obtener mejor nivel de compresión respecto de los anteriores. De aquí en adelante nos ocuparemos de esta categoría.

Los compresores específicos de XML se pueden clasificar según diferentes criterios.

Decimos que un documento XML está "bien formado" si su estructura sintáctica básica se compone de elementos, atributos y comentarios como se especifica en la definición de XML. Ahora bien, cada aplicación de XML (es decir, cada lenguaje definido con esta tecnología) puede definir cuál es la relación que debe verificarse entre los distintos elementos presentes en el documento.

Esta relación entre elementos se especifica en un documento externo o definición, expresada como *Definición de Tipo de Documento*[1] (DTD, Document Type Definition) o como *XSchema*[2]. Crear una definición equivale a crear un nuevo lenguaje de markup, para una aplicación específica.

Entonces, según la dependencia de dicha información de esquema tenemos compresores:

- **Dependientes de esquema**

  Tanto compresor como descompresor deben tener acceso a la información de esquema del documento para completar su proceso.

- **Independientes de esquema**

  La información de esquema no es necesaria para comprimir ni descomprimir.

Aunque en teoría los compresores que utilizan la información de esquema logran alguna mejora en el nivel de compresión, en la práctica no son tan utilizados ya que no siempre se cuenta con dicha información.

Otra clasificación posible es según la posibilidad de efectuar consultas (*queries*) sobre el documento comprimido. Existen extensiones a XML que proveen formas de acceder, manipular y devolver XML. Las dos más importantes son *XPath*[3] y *XQuery*[4]. XPath permite referirse a secciones individuales de un documento XML, y dar acceso aleatorio a los datos a otras tecnologías o aplicaciones. Por otro lado, XQuery es a XML lo que el SQL a las bases de datos relacionales, un lenguaje diseñado para consultar colecciones de datos en XML.

- **No consultables** (*Non-queriable*)

  No permiten procesar consultas en el archivo comprimido. El objetivo suele ser lograr el mayor nivel de compresión posible (en esta clase se podría incluir también a los compresores de texto general).

- **Consultables** (*Queriable*)

  Se permiten consultas, aunque en este caso el nivel de compresión suele ser inferior al de los citados arriba. El objetivo es evitar la descompresión completa del archivo durante la ejecución de una consulta. Esta característica es importante para aplicaciones que corren en dispositivos con recursos limitados (dispositivos móviles, sistemas de GPS).

[1] `http://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm`
[2] `http://www.w3.org/XML/Schema`
[3] `http://www.w3.org/TR/xpath20/`
[4] `http://www.w3.org/XML/Query`

Una tercera categorización se basa en la capacidad del algoritmo de compresión/descompresión de operar en línea (*online*) o no (*offline*):

- **Online**

  El descompresor puede tomar el flujo de salida del compresor y procesarlo en línea, es decir antes de que éste termine de transmitir la totalidad de los datos comprimidos.

- **Offline**

  Es necesario contar con el archivo comprimido completo para realizar el proceso de descompresión.

La característica de ser *online* puede resultar importante en escenarios en los que los usuarios intercambian mucha información XML en una red, disminuyendo los tiempos de latencia durante el proceso de transmisión.

2.2. Características deseables

En esta sección se presentan algunas características generales que se podrían evaluar de un compresor específico de XML y qué detalles considerar.

- **Compresión efectiva**

  El objetivo es lograr un buen nivel de compresión, minimizando tiempos de compresión/descompresión y el consumo de memoria. En la práctica se logran mejores niveles de compresión si se explotan la redundancia en la estructura del documento XML y las características comunes en los datos. Para optimizar los tiempos se espera que el documento XML se parsee y procese en una pasada. Por otro lado, un documento XML puede ser muy grande haciendo imposible cargarlo en su totalidad en memoria para comprimir/descomprimir, y entonces debería haber un límite en el tamaño de la ventana de memoria utilizada para dichos procesos. Por estas razones un parser SAX (ver recuadro) suele ser la mejor opción.

SAX vs DOM

Existen 2 APIs principales para recorrer y acceder a los datos de documentos XML:

- **Tree-based APIs**

  Mapean el documento XML en una estructura interna de árbol, y luego permiten navegar ese árbol. El grupo de trabajo Document Object Model (DOM) en W3C mantiene una API recomendada, y hay variantes de otras fuentes.

- **Event-based APIs**

  Reportan los eventos del parsing (tales como el comienzo y fin de elementos) directamente a la aplicación a través de callbacks. La aplicación implementa manejadores para los diferentes eventos. SAX es el ejemplo más conocido de esta clase.

Las APIs tree-based son útiles para un amplio rango de aplicaciones, pero en general requieren mayores recursos del sistema, en especial si el documento es grande. Por otro lado, muchas aplicaciones necesitan crear sus propias estructuras de datos además del árbol provisto por la API, siendo ineficiente construir un árbol sólo para mapear sus nodos en nuevas estructuras y luego descartarlo.
En esos casos una API event-based provee un acceso simple y de más bajo nivel al documento XML: se pueden parsear documentos mucho más grandes que la memoria disponible en el sistema, y crear las estructuras de datos propias a partir de los manejadores de eventos.

- **Lenguaje de consulta expresivo y consultas eficientes**

  Idealmente, el compresor debería soportar un lenguaje de consulta expresivo, como así también una evaluación eficiente del mismo. *XPath*[5] y *XQuery*[6] son los lenguajes de consulta más comunes, además de ser estándares propuestos por W3C. Uno de los problemas de efectuar consultas sobre los datos comprimidos es el costo de la descompresión que sea necesaria, pero existen alternativas para que el impacto sea menor (como se verá más adelante).

- **Mínima intervención humana y mínimo uso de estructuras auxiliares**

  El compresor debería ser lo suficientemente general, es decir contar con un nivel mínimo de información adicional requerida (como DTD, o XSchema, o determinación de compresores especializados según el tipo de datos para mejorar el nivel de compresión). Por otro lado, ya sea para mejorar la compresión o para procesar consultas más eficientemente, se usan estructuras auxiliares; lo que no hay que dejar de tener en cuenta es que éstas deberían ser pensadas estratégicamente, ya que contribuyen al tamaño del archivo comprimido y a los tiempos de procesamiento.

Como se puede apreciar, estos criterios pueden entrar en conflicto entre sí. Por ejemplo, si se decide comprimir los datos en bloques se suele obtener un mejor nivel de compresión, pero para realizar consultas es necesario descomprimir el archivo completo. Por otro lado, si se comprimen los ítems de datos indivualmente se puede evitar descomprimir todo al consultar, pero se degradan el nivel de compresión y los tiempos.

[5] `http://www.w3.org/TR/xpath`
[6] `http://www.w3.org/XML/Query/`

Como se menciona antes, las estructuras auxiliares pueden ayudar a mejorar la eficiencia de las consultas, pero a la vez incrementan el tamaño del archivo y el tiempo de procesamiento al necesitar construir y cargar dichas estructuras en memoria. En general, para obtener un mayor nivel de compresión se requiere mayor tiempo en el proceso de compresión.

Alrededor de estos desafíos se han ido desarrollando nuevas alternativas que tienden a acercarse a estas cualidades, o que priorizan alguna de ellas de acuerdo a la finalidad con las que se las va a emplear en la práctica. A lo largo de los próximos capítulos estudiaremos las distintas maneras utilizadas para aproximarse a estas propiedades descriptas, implementaciones particulares y ejemplos de casos de uso.

Capítulo 3

Técnicas de Compresión para Documentos XML

A lo largo de este capítulo se introducen de forma general las diferentes técnicas y métodos utilizados en la práctica para la compresión de documentos XML, tanto para el caso de compresores que no permiten consultas como para aquellos que sí. En el capítulo siguiente se describen las variantes particulares de las técnicas expuestas aquí, implementadas por compresores de XML existentes.

3.1. Separación de la estructura y los datos

La idea es comprimir por separado la estructura, representada por los tags y atributos de XML, y los datos en sí. En general, agrupar los valores de un mismo elemento implica una mejor compresión usando algoritmos de compresión de texto basados en diccionario[1] y ventana deslizante (*LZ77, LZ78*)[2], ya que es esperable que dichos valores formen patrones que se repitan, y entonces haya más coincidencias dentro de la ventana de compresión.

La estructura se suele codificar de tal forma de mantener información acerca de los tags y sus posiciones (comienzo y final), preservando la información de la jerarquía. Un esquema común consiste en asignar números enteros a los tags y atributos, y algún identificador para señalar el cierre de los primeros.

Los contenedores (los grupos de valores de cada elemento) se pueden comprimir de forma separada o concatenándolos en un único flujo de datos, en la mayoría de los casos utilizando algoritmos de compresión basados en diccionario.

[1] `http://en.wikipedia.org/wiki/Dictionary_coder`
[2] `http://en.wikipedia.org/wiki/LZ77_and_LZ78`

3.2. Codificación de datos numéricos

Una técnica utilizada para comprimir secuencias de dígitos (o incluso 1 dígito) es reemplazar cada ocurrencia de ellas por una bandera en el flujo principal, mientras que el valor en sí se ubica en un contenedor aparte, codificado como un número en base 256. Distintos resultados pueden obtenerse cambiando dicha base, aunque usar la codificación con mayor densidad posible parece ser la mejor opción en promedio.

La bandera consiste, por ejemplo, de un único caracter, entre '1' y '4', que identifica el largo en bytes del valor. Si una secuencia representa un valor mayor que 256^4 (entero más largo en 4 bytes), se divide en secuencias más cortas.

Algunos datos numéricos representan tipos de información específica, como ser una fecha. Para estos casos, si se tuviera por ejemplo AAAA-MM-DD (año, mes, día), usando el esquema descripto se necesitarían 5 bytes (incluyendo los '-') en el flujo principal, más 4 bytes en el contenedor de números para codificar este valor.

Asumiendo que cada mes tiene hasta 31 días, todas las fechas en el intervalo 1977-01-01 a 2153-02-26 se pueden representar como un entero en el rango 0-65535. Así una fecha se podría codificar en sólo 3 bytes: una bandera en el flujo principal, y un entero de 2 bytes en el contenedor. Enfoques similares se adoptan para distintos casos (este tipo de estrategias se suele ubicar dentro de los compresores semánticos).

3.3. Compresores semánticos

Como hemos visto, una técnica común es agrupar los valores de cada elemento en distintos contenedores. La idea en este caso consiste en definir algoritmos de compresión especializados para cada contenedor, ya sean compresores predefinidos para los tipos básicos, compresores combinados o definidos por el usuario.

Una de las limitaciones de este enfoque es que puede ser necesaria la intervención del usuario para indicar qué compresor/descompresor usar para un elemento dado. Es por eso que existen investigaciones que buscan diseñar motores de inferencia de tipos, y así evitar la necesidad de intervención humana para determinar el tipo de datos. Dependiendo del tipo inferido se aplican los métodos de codificación apropiados, de forma tal de obtener mayor nivel de compresión y menor carga para el usuario.

3.4. Uso de DTD

No existen muchos algoritmos que hagan uso de la información contenida en un DTD (o XSchema). Los métodos propuestos se basan en comprimir sólo la información del documento XML que no está contenida en el DTD. Tomando en cuenta que la sintaxis del documento ya está en el DTD, el nivel de compresión obtenido puede mejorar.

Como ejemplos generales de uso de la información en un DTD para la compresión se pueden nombrar: la reutilización de los elementos, atributos y símbolos ya definidos en el DTD, predecir los tags de apertura o cierre de los elementos, o sistemas para codificar la estructura del documento.

La principal desventaja de estos compresores es la necesidad de contar con el DTD del documento, ya sea para comprimir o descomprimir (en cuyo caso debemos contar con exactamente el mismo DTD usado al comprimir). En la práctica muchas veces uno no tiene (o incluso puede no estar definido) el DTD, o la estructura de un documento es susceptible de cambios en su organización en el tiempo, produciendo incompatibilidades entre compresor y descompresor.

3.5. Otras técnicas

Existen varias propuestas y algunos desarrollos que hacen uso de la observación de que una estructura válida de un documento XML se puede describir a partir de una *gramática libre de contexto*[3], y entonces aplicar técnicas de compresión basadas en gramáticas (denominadas también de compresión sintáctica). Se trata de una generalización de las técnicas basadas en diccionario. Aunque interesante, esta aproximación no ha dado origen todavía a compresores competitivos.

Existen dos alternativas principales usadas para compresión sintáctica. Una de ellas fija una gramática G, conocida tanto por el compresor como por el descompresor, tal que el lenguaje generado por G contiene todas las cadenas a comprimir. Así, para comprimir una cadena particular, se comprime su árbol de derivación. La otra alternativa, asigna a cada cadena x una gramática diferente G_x, de tal manera que el lenguaje generado por G_x es $\{x\}$. Si la cadena a comprimir es x, el codificador transmite la información necesaria al decodificador para reconstruir la gramática G_x.

Por otro lado, también se encuentran alternativas basadas en modelar de manera probabilística la estructura de un documento XML. Los datos en un documento contienen mucha información redundante, presente en la estructura. Para los compresores comunes esta redundancia es difícil de descubrir, siendo esta la razón de resultados no óptimos. En el caso de encontrar y explotar esa redundancia, uno podría mejorar la eficiencia de la compresión.

3.6. Técnicas en compresores consultables

La solución trivial para que sea posible efectuar consultas a un documento XML comprimido es la de comprimir los ítems de datos individualmente, para no tener que descomprimir el documento al evaluar un pedido. Sin embargo, de esta manera no se obtiene un buen nivel de compresión, como es de suponer.

Una variante para lograr mejores resultados es utilizar una transformación homomórfica. Esta estrategia consiste en comprimir el documento XML mediante una transformación que preserve la

[3] `http://en.wikipedia.org/wiki/Context-free_grammar`

estructura sintáctica y la información del documento original [ver 4.6, 4.7]. Esto implica además que el documento comprimido puede ser parseado como cualquier otro documento XML, usando un parser SAX o DOM, pero sin necesidad de descompresión.

Otras alternativas construyen estructuras de datos auxiliares para poder agrupar los valores comprimidos en contenedores, o comprimir por bloques, manteniendo punteros a los respectivos nodos [ver 4.8, 4.9]; o estructuras para llevar los caminos posibles en el documento, y punteros de valores comprimidos a caminos (restringiendo posiblemente las consultas permitidas) [ver 4.8].

Capítulo 4

Compresores de XML

En este capítulo se presentan distintas herramientas existentes para la compresión de documentos XML, como así también las técnicas utilizadas por cada una y las características propias de cada implementación.

4.1. XMill

Url: `http://sourceforge.net/projects/xmill/`
Licencia: BSD, GPL
Versión disponible: 0.8 (Marzo de 2003)
Descripción: Incluida junto con el código fuente
Clasificación: Específico XML - Independiente de esquema - No consultable - Offline

XMill aplica tres principios para comprimir XML:

- Separación de estructura y datos

  La estructura consiste de tags y atributos, que forman un árbol. Los datos son una secuencia de ítems (cadenas de texto) que representan los contenidos de los elementos y los valores de los atributos. Estructura y datos se comprimen por separado.

- Agrupamiento de ítems con significado relacionado

  Los datos se agrupan en contenedores por elemento y cada contenedor se comprime por separado.

- Empleo de compresores semánticos

  Algunos datos son texto, otros son números, otros secuencias de ADN (o cualquier otro tipo de información específica de un campo particular). XMill permite aplicar compresores especializados a diferentes elementos, ya sea mediante alguno de los compresores incluidos o alguno implementado por el usuario siguiendo las instrucciones provistas.

Un parser SAX recorre el archivo, enviando los identificadores de lo que va encontrando al módulo principal de XMill (llamado *path processor*). A cada elemento se le asigna un contenedor. Tags y atributos se envían al contenedor para la estructura, mientras que los datos se envían a distintos contenedores de acuerdo al elemento al que pertenecen (aunque también el usuario puede decidir cómo agruparlos). Luego esos contenedores se comprimen por separado.

```
1 <book>
2   <title lang="en">Refactoring</title>
3   <author>Fowler</author>
4   <author>Beck</author>
5 </book>
```

Listing 4.1: Fragmento de documento XML

El fragmento del ejemplo 4.1 se codificaría de la siguiente manera según XMill:

```
T1 T2 T3 C3 / C4 / T4 C5 / T4 C5 / /
```

a lo que se suma un diccionario que relaciona los tags (y atributos) y su codificación: book = T1, title = T2, @lang = T3, author = T4. A los datos se les asignan los contenedores C3, C4 y C5, dependiendo del tag padre.

Los contenedores se mantienen en memoria en una ventana de tamaño fijo (por defecto 8MB). Cuando esta ventana se llena, todos los contenedores se comprimen con *gzip* (existe la opción de usar *bzip2* y *ppm*, también), se guardan en disco, y luego la compresión continúa. Esto divide el archivo de entrada en varios bloques comprimidos independientes.

Después de cargar y de-zipear (*gunzip*, o el respectivo descompresor general) los contenedores, el descompresor (*Xdemill*) parsea el contenedor de la estructura, invocando al correspondiente descompresor semántico para los datos y generando la salida. Opcionalmente, XMill puede preservar los espacios en blanco, para lo cual crea un contenedor extra.

4.2. XComp

Url: No disponible (experimental, no disponible públicamente)
Licencia: No especificada
Versión disponible: -
Descripción: `http://se.uwaterloo.ca/~ddbms/publications/distdb/Weimin.pdf`
Clasificación: Específico XML - Independiente de esquema - No consultable - Offline

Los principios detrás de XComp son muy similares a los de XMill. De hecho tan sólo agrega algunas variantes. En este caso la estructura del documento XML se codifica como una secuencia de enteros, mientras que los datos se agrupan de acuerdo a los elementos a los que pertenecen y su nivel en el árbol del documento (es decir que si tenemos un mismo tag en distinto niveles del árbol, se le asignarán distintos contenedores).

En mayor detalle, este compresor asigna un entero positivo comenzando en 10 como identificador de cada tag y atributo; los enteros hasta 10 están reservados con distintos significados.

Para el ejemplo presentado anteriormente se tendría: book = 10, title = 11, lang = 12, author = 13. Por otro lado, el 0 indica el cierre de un tag, y el 1 indica la posición de un dato. De esta manera la estructura de nuestro ejemplo 4.1 quedaría:

```
10, 11, 12, 1, 0, 13, 1, 0, 13, 1, 0, 0
```

Notar que no hay un 1 después de 12 para señalar la posición del valor del atributo, ya que un atributo sólo puede tener un único ítem.

Hasta aquí se mantienen las posiciones de los datos, que luego serán agrupados y comprimidos por separado. Para poder decodificarlos más tarde, se registran sus longitudes. Para el caso del ejemplo:

```
11, 6, 4
```

Otro esquema propone reemplazar los 1 de la estructura por los largos de los datos (distinguiéndolos de alguna manera del resto de los identificadores).

Opcionalmente existen alternativas para preservar los espacios en blanco también, asignando un identificador y la longitud de los sectores de esas características.

4.3. XWRT

Url: `http://xwrt.sourceforge.net/`
Licencia: GPL
Versión disponible: 3.2 (Octubre de 2007)
Descripción: `http://www.ii.uni.wroc.pl/~inikep/papers/07-AsymmetricXML.pdf`
Clasificación: Específico XML - Independiente de esquema - No consultable - Offline

XWRT (*XML Word Replacing Transform*), además de las ideas discutidas en los compresores anteriores (separación de estructura y datos, agrupamiento en contenedores) incorpora otras variantes: codificación específica para números y fechas (como se explicó en el capítulo anterior) y una técnica de compresión basada en diccionarios. La idea consiste en reemplazar las palabras que aparecen frecuentemente por referencias al diccionario que se construye en una pasada previa sobre los datos.

Después del preprocesamiento de los datos, la codificación resultante se pasa a alguno de los siguientes compresores de propósito general: *gzip* (el más usado), *LZMA*[1] o *PPM*[2].

[1] `http://en.wikipedia.org/wiki/LZMA`
[2] `http://en.wikipedia.org/wiki/Prediction_by_Partial_Matching`

Como se menciona antes, el principal cambio respecto de los algoritmos ya descriptos es el reemplazo de palabras frecuentes con referencias a un diccionario; claro que la noción de palabra en este contexto es más amplia: se consideran secuencias de una longitud mínima dada y que aparezcan en el documento una cierta cantidad mínima de veces.

El diccionario contiene ítems de las siguientes clases:

- Palabras comunes
- Tags de apertura
- Prefijos de URLs (por ejemplo de la forma http://domain/)
- Direcciones de correo electrónico
- Entidades XML (por ejemplo &)
- Secuencias de espacios

Las palabras del diccionario se escriben explícitamente, con separadores, al comienzo del archivo de salida. Las referencias al diccionario se codifican usando un prefijo (un byte especial) predeterminado.

4.4. eXalt

Url: `http://exalt.sourceforge.net/`
Licencia: GPL
Versión disponible: 0.1.0 (Febrero de 2003)
Descripción: `http://exalt.sourceforge.net/thesis.pdf`
Clasificación: Específico XML - Independiente de esquema - No consultable - Offline

eXalt (*An Experimental XML Archiving Library/Toolkit*) toma un punto de vista diferente y, considerando que un documento XML puede representarse usando una gramática libre de contexto, aplica una técnica (introducida por Kiefer y Yang[3]) para codificar gramáticas (compresión sintáctica). La idea es encontrar alguna forma de modelado del documento que ayude a descubrir la redundancia. eXalt implementa dos estrategias: modelado simple, y modelado adaptativo.

La técnica de modelado simple transforma el XML de entrada de tal manera que las ocurrencias de los elementos y atributos conocidos, así como los cierres de los elementos, sean representados por identificadores numéricos; luego se aplica la compresión sintáctica. En este caso se trata de una metodología similar a la usada en las implementaciones anteriores.

La estrategia de modelado adaptativo intenta hacer un modelado probabilístico de los elementos. Mientras se lee el XML de entrada se construyen modelos para cada elemento que reflejen

[3] `http://www.ims.cuhk.edu.hk/~cis/2002.1/KIEFFER.pdf`

su estructura y características probabilísticas. Estos modelos se usan luego para predecir la estructura de los datos siguientes y mejorar la performance de la compresión.

Hay que notar que esta última no siempre es mejor que la estrategia simple. De hecho, son muy similares en el caso promedio; sin embargo el modelado adaptativo es más rápido, ya que el compresor sintáctico tiene que procesar menos símbolos.

Una característica particular de eXalt es que puede funcionar como un parser SAX, es decir, leer los datos comprimidos de un documento XML y emitir los eventos SAX.

4.5. XMLPPM

Url: `http://xmlppm.sourceforge.net`
Licencia: GPL
Versión disponible: 0.98.3 (Febrero de 2008)
Descripción: `http://xmlppm.sourceforge.net/paper/paper.html`
Clasificación: Específico XML - Independiente de esquema - No consultable - Online

XMLPPM es una adaptación del esquema de compresión PPM (*Prediction by Partial Matching, Predicción por Coincidencia Parcial*). Los modelos PPM mantienen estadísticas de qué símbolos han sido vistos en el contexto de símbolos anteriores. El modelo se usa para estimar un rango de probabilidad y usar codificación aritmética[4] para cada símbolo. Mientras, el modelo se va actualizando con cada nuevo símbolo leído.

La estimación de la probabilidad se hace de la siguiente manera: si el símbolo ha sido visto en el contexto coincidente más largo, entonces la probabilidad es la frecuencia relativa en ese contexto; caso contrario, se codifica un símbolo de escape y se prosigue con el siguiente contexto. El decodificador mantiene el mismo modelo, y usa los símbolos que va decodificando para ir actualizándolo.

XMLPPM usa esta idea con algunas modificaciones, para permitir backtracking en la estructura del documento y usar contextos jerárquicos basados en el camino de la raíz al símbolo. Por ejemplo, al codificar "$< a >< b > X < /b > Y < /a >$", se usa "$< a >< b >$" como contexto de X, y usa "$< a >$" como contexto de Y.

El compresor usa cuatro modelos: uno para nombre de elementos y atributos, uno para la estructura de los elementos, uno para los valores de los atributos y otro para las demás cadenas de texto. Cada modelo mantiene su propio estado, pero comparten el acceso al codificador aritmético.

4.6. XGrind

Url: `http://xgrind.sourceforge.net/`
Licencia: GPL

[4] `http://en.wikipedia.org/wiki/Arithmetic_encoding`

Versión disponible: - (Julio de 2002)
Descripción: `http://reference.kfupm.edu.sa/content/x/g/xgrind__a_query_friendly_xml_compressor__87177.pdf`
Clasificación: Específico XML - Dependiente de esquema - Consultable - Online

Un documento XML comprimido con XGrind se genera de la siguiente forma: en primer lugar se crea un diccionario para almacenar todos los nombres de elementos y atributos (en una primera pasada sobre el documento, o desde su DTD). En la primera pasada también se recolectan estadísticas para crear modelos para compresión mediante *codificadores de Huffman*[5].

Una vez que los codificadores de Huffman se crearon, XGrind inicia el proceso de compresión, que requiere una nueva lectura del documento. El compresor identifica los nombres de tags y atributos mediante índices que referencian a las correspondientes entradas del diccionario construido en el paso anterior. Los datos de elementos y atributos se comprimen individualmente mediante Huffman.

Después de completar este paso, el documento comprimido y las estructuras auxiliares (diccionario, tablas de los codificadores de Huffman) se concatenan como salida.

Como este proceso es una transformación homomórfica, todas las operaciones que se podían ejecutar sobre el documento original, como por ejemplo las consultas, son posibles en el documento comprimido. De hecho se pueden utilizar las herramientas y técnicas existentes con algunas modificaciones. Además, las consultas por coincidencia exacta se pueden realizar sin descomprimir el documento (ya que se puede transformar el valor de la consulta usando el correspondiente valor codificado por Huffman para efectuar la comparación durante la búsqueda).

4.7. XPress

Url: No disponible (experimental, no disponible públicamente)
Licencia: No especificada
Versión disponible: -
Descripción: `http://islab.kaist.ac.kr/chungcw/InterConfPapers/sigmod2003_jkmin.pdf`
Clasificación: Específico XML - Independiente de esquema - Consultable - Online

XPress se basa en un esquema de codificación llamado *Reverse Arithmetic Encoding* (Codificación Aritmética Inversa). Esta técnica está diseñada para codificar los caminos en el árbol de un documento XML usando intervalos de números reales. Cada intervalo se encuentra en el rango que va entre 0.0 (inclusive) y 1.0 (no incluido).

Estos intervalos satisfacen la propiedad de contención de sufijos. Esta propiedad asegura que si el camino P a un elemento es sufijo de un camino Q a otro elemento, entonces el intervalo que representa a P, denotado por I_P, debe contener al intervalo que representa a Q, I_Q.

[5] `http://en.wikipedia.org/wiki/Huffman_coding`

Por ejemplo, si se tuviera el camino "//journal/title", su intervalo contendría al intervalo de "/paper/entry/journal/title".

De esta manera, XPress mejora la estrategia de compresión de XGrind en dos aspectos. En lugar de sólo codificar los tags del documento, también codifica los caminos a los elementos usando intervalos de números reales que cumplen la propiedad de contención de sufijos. Esto permite evaluar consultas basadas en camino sobre el documento comprimido verificando la contención de los intervalos de la consulta y de los elementos, sin descomprimir el documento. Por otro lado, los valores numéricos se codifican preservando orden, permitiendo además de consultar por coincidencia exacta, hacerlo por rangos, sin necesidad de descomprimir los valores.

4.8. XQueC

Url: `http://staff.icar.cnr.it/angela/xquec/` (prototipo, no disponible públicamente aún)
Licencia: No especificada
Versión disponible: -
Descripción: `http://www-rocq.inria.fr/~manolesc/PAPERS/XQueCTOITrevised.pdf`
Clasificación: Específico XML - Independiente de esquema - Consultable - Online

XQueC separa la estructura del documento XML de los datos. Los ítems de dato que comparten el mismo camino raíz-hoja se agrupan en un mismo contenedor. Para proveer un procesamiento de consultas más eficiente, construye además la estructura de árbol del documento y un listado donde mantiene todos los distintos caminos en el documento.

Para permitir un acceso eficiente a los datos, XQueC referencia cada ítem individualmente comprimido al correspondiente nodo en el árbol, y cada contenedor al correspondiente camino en el listado. Estas estructuras auxiliares pueden derivar en menores niveles de compresión en comparación a otros compresores.

4.9. XQzip

Url: No disponible (experimental, no disponible públicamente)
Licencia: No especificada
Versión disponible: -
Descripción: `http://www.cais.ntu.edu.sg/~jamescheng/XQzip_edbt04.pdf`
Clasificación: Específico XML - Independiente de esquema - Consultable - Online

XQzip soporta varias consultas XPath sobre los datos comprimidos a partir de su esquema de *Structural Indexing Tree* (SIT), o Árbol de Estructura de Indexado. Este árbol se construye en base al particionamiento de los caminos equivalentes en el documento XML. Los datos y el SIT se comprimen en distintos contenedores, que a su vez se subdividen en bloques más pequeños

que se pueden comprimir/descomprimir como unidades individuales. De esta manera se busca un equilibrio entre el nivel de compresión y la necesidad de descompresión al procesar consultas (explota la repetición en los datos, y evita la descompresión completa para evaluar consultas).

Otra característica es el empleo de un *buffer* para evitar la descompresión reiterada cuando se evalúan consultas si los bloques de datos ya están en el buffer. Con el uso del SIT y el buffer recién descripto, XQzip alcanza buenos tiempos de consulta. Sin embargo, es difícil determinar un tamaño de bloque para lograr un balance óptimo. Otra limitación es que el SIT no soporta consultas complejas.

Capítulo 5

Evaluación de Compresores

En esta sección se presenta una serie de criterios para evaluar y comparar las distintas alternativas existentes y disponibles para comprimir documentos XML. Luego se describe el entorno en el que se realizaron las pruebas, así como el corpus de documentos utilizado. Finalmente se exponen los resultados del experimento.

5.1. Criterios

Para evaluar la performance de las implementaciones consideradas se tomaron las siguientes métricas:

- **Nivel de compresión** (*Compression Ratio*)

  Existen dos expresiones diferentes usadas comúnmente como *Compression Ratio* (CR). Una, que llamaremos *CR1*, expresa el número de bits en el documento comprimido requeridos para representar un byte del documento original. En este caso, cuanto menor sea el valor, mejor nivel de compresión. Por otro lado, la otra métrica, que denotaremos por *CR2*, expresa la fracción (en porcentaje) en que el documento de entrada se reduce. Es decir que el nivel de compresión será mayor cuanto más alto sea el porcentaje de CR2.

  CR1 = (tamaño de archivo comprimido * 8 / tamaño de archivo original) bits/byte

  CR2 = [1 - (tamaño de archivo comprimido / tamaño de archivo original)] * 100

  La principal diferencia entre estas métricas es que CR1 es proporcional al tamaño del espacio efectivamente requerido para guardar el archivo comprimido, mientras que CR2 muestra la reducción en porcentaje del tamaño del archivo comprimido respecto del original. La noción detrás de CR1 nos da la intuición de la *cantidad de información* necesaria para representar un byte, una medida usual en teoría de la información.

- **Tiempo de compresión** (*Compression Time*)

Representa el tiempo que lleva el proceso de compresión, es decir, el período entre el inicio de la ejecución del programa y el momento en que los datos comprimidos fueron escritos en el archivo de destino.

- **Tiempo de descompresión** (*Decompression Time*)

  Representa el tiempo que lleva el proceso de descompresión, es decir, el período entre el inicio de la ejecución del programa y el momento en que los datos comprimidos fueron restaurados y se obtuvo el documento original.

En el caso de los tiempos de compresión y descompresión, vale mencionar que los valores obtenidos son relativos a la máquina y entorno en que se realizaron las pruebas, pero sirven como medida para hacer la comparación relativa entre compresores.

5.2. Compresores evaluados

Se evaluaron compresores de texto general como referencia (*gzip*, *bzip2*), y compresores específicos para XML; dentro de estos últimos se tuvieron en cuenta aquellos cuyas implementaciones (y código fuente) están disponibles, compilan y corren en entornos Linux. Estas condiciones limitaron las alternativas, y dejaron afuera a los compresores consultables.

A pesar de que no fue posible compilar XMill en Linux, se decidió tomar los resultados de nivel de compresión para la versión binaria en Windows, ya que este compresor es un referente dentro de los específicos para documentos XML, tanto por ser uno de los primeros como por las técnicas que emplea. Por supuesto que en este caso no se consideraron los tiempos de compresión/descompresión, aunque en la práctica se trata de uno de los compresores específicos más veloces, y sus tiempos se aproximan a los de gzip según distintas referencias [Sak08] [WNC06] [LS00].

Entonces se consideraron:

- **gzip**
  Version: 1.3.12 (Abril de 2007)
  Download: `ftp://wuarchive.wustl.edu/mirrors/gnu/gzip/gzip-1.2.4.tar.gz`
- **bzip2**
  Version: 1.0.5 (Marzo de 2008)
  Download: `http://www.bzip.org/1.0.5/bzip2-1.0.5.tar.gz`
- **XMill**
  Version: 0.8 (Marzo de 2003)
  Download: `http://sourceforge.net/projects/xmill/files/xmill/xmill-0.8.zip`
- **XWRT**
  Version: 3.2 (Octubre de 2007)
  Download: `http://sourceforge.net/projects/xwrt/files/xwrt/XWRT32.zip`

- **XMLPPM**
  Version: 0.98.3 (Febrero de 2008)
  Download: `http://sourceforge.net/projects/xmlppm/files/xmlppm-src/xmlppm-0.98.3.tar.gz`

5.3. Evaluación

5.3.1. Entorno

La evaluación se efectuó en un entorno Linux (Ubuntu 9.04) con kernel 2.6.27, corriendo sobre un CPU Intel(R) Pentium(R) 4 HT CPU 2.40GHz con 1GB de memoria RAM (dual channel). Como complemento de la información recogida en este experimento se sugiere ver [Sak08] y *Benchmark of XML Compression Tools* (`http://xmlcompbench.sourceforge.net/`), que refleja los resultados de experimentos similares sobre un corpus de 57 documentos en dos plataformas diferentes (una de recursos computacionales elevados y otra limitada).

Para recabar los datos se utilizó un script de automatización que puede consultarse en el apéndice A. El mismo es multiplataforma, y almacena en un archivo de valores separados por coma los resultados de las distintas ejecuciones para cada compresor, y para cada dataset.

Una vez obtenidos esos datos, se procesaron utlizando una planilla de cálculos con la cual se determinaron los promedios y se confeccionaron los gráficos.

5.3.2. Datasets

Para la experiencia se utilizaron cinco fuentes de datos, cubriendo distintos formatos y estructuras de documentos XML:

- **Wikipedia-Group**

  Wikipedia[1] ofrece copias de sus contenidos a los usuarios interesados. Se tomaron 2 muestras de archivos de volcado de datos con diferentes tamaños y características.

- **Shakespeare**[2]

  Representa una colección de obras de Shakespeare, etiquetadas. Contiene muchas secciones textuales extensas.

- **SwissProt**[3]

  Se trata de una base de datos de secuencias de proteínas que describen cadenas de ADN. Tiene un alto nivel de anotaciones y baja redundancia.

[1] `http://download.wikimedia.org/`
[2] `http://www.cs.wisc.edu/niagara/data/shakes/shaksper.htm`
[3] `http://ca.expasy.org/sprot/`

- **Treebank**[4]

  Es una colección de oraciones etiquetadas tomadas del Wall Street Journal. Tiene mucha profundidad de elementos, y una estructura recursiva pero no regular.

- **XMark-Group**

  Estos documentos modelan una base de datos de subastas, con un alto nivel de profundidad de elementos que mantienen cierta regularidad. Las instancias de estos documentos se generan a partir de la herramienta *xmlgen*[5]. Para el experimento se generaron 2 variantes.

Dataset	Documento	Tamaño (MB)	Tags	Número de nodos	Profundidad
Wikipedia	EnWikiNews.xml	71.09	20	2013778	5
	EnWikiQuote.xml	127.25	20	2672870	5
Shakespeare	Shakespeare.xml	7.47	22	574156	7
SwissProt	SwissProt.xml	112.13	85	13917441	5
Treebank	Treebank.xml	84.06	250	10795711	36
XMark	XMark1.xml	11.40	74	520546	12
	XMark2.xml	113.80	74	520546	12

5.3.3. Resultados

Para cada combinación de documento y compresor se realizaron corridas de compresión/descompresión utilizando los parámetros por defecto. Para tener una aproximación más precisa a la hora de reportar los tiempos, se tomó el promedio de 5 ejecuciones de las que se descartaron los tiempos en los extremos inferior y superior.

A continuación se presenta una serie de tablas y gráficos con los resultados obtenidos.

Compression Ratio 1

Dataset	Documento	gzip	bzip2	xmill	xmlppm	xwrt
Wikipedia	EnWikiNews.xml	2.28	1.69	0.67	1.51	1.59
	EnWikiQuote.xml	2.77	2.13	1.72	1.95	2.03
Shakespeare	Shakespeare.xml	2.18	1.49	1.60	1.37	1.48
SwissProt	SwissProt.xml	0.97	0.59	0.39	0.45	0.46
Treebank	Treebank.xml	2.9	2.41	2.20	2.34	2.65
XMark	XMark1.xml	2.57	1.73	1.74	1.58	1.50
	XMark2.xml	2.58	1.73	1.72	1.58	1.43
	Promedio	2.32	1.68	1.43	1.54	1.59

[4] `http://xmlcompbench.sourceforge.net/Treebank.rar`
[5] `http://monetdb.cwi.nl/xml/generator.html`

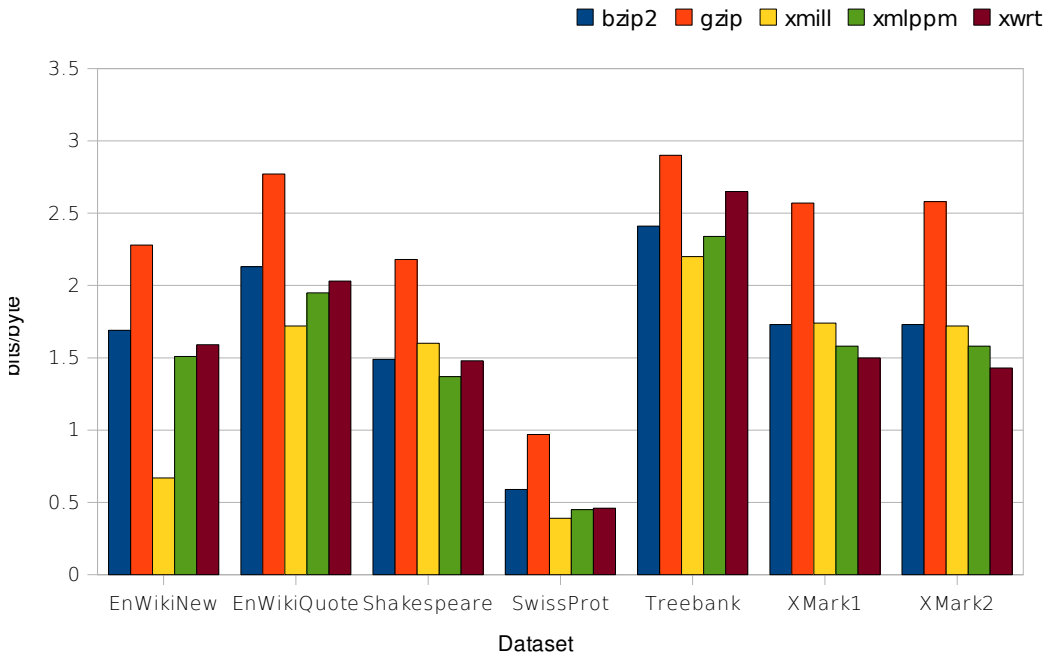

FIGURA 5.1: Compression Ratio 1 (promedio por dataset)

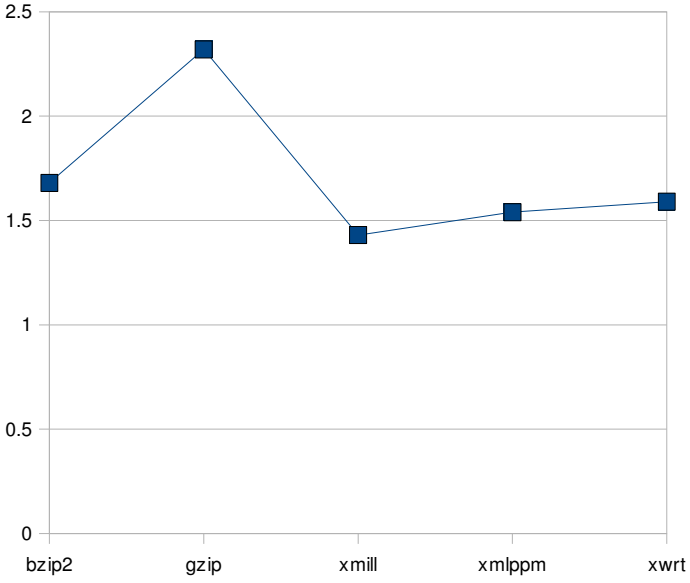

FIGURA 5.2: Compression Ratio 1 (promedio)

Compression Ratio 2

Dataset	Documento	gzip	bzip2	xmill	xmlppm	xwrt
Wikipedia	EnWikiNews.xml	71.53	78.91	91.63	81.16	80.09
	EnWikiQuote.xml	65.43	73.34	78.52	75.65	74.60
Shakespeare	Shakespeare.xml	72.72	81.36	79.94	82.91	81.48
SwissProt	SwissProt.xml	87.93	92.60	95.18	94.35	94.23
Treebank	Treebank.xml	63.74	69.89	72.55	70.76	66.91
XMark	XMark1.xml	67.82	78.35	78.21	80.23	81.31
	XMark2.xml	67.73	78.42	78.47	80.28	82.13
	Promedio	70.99	78.98	82.07	80.76	80.11

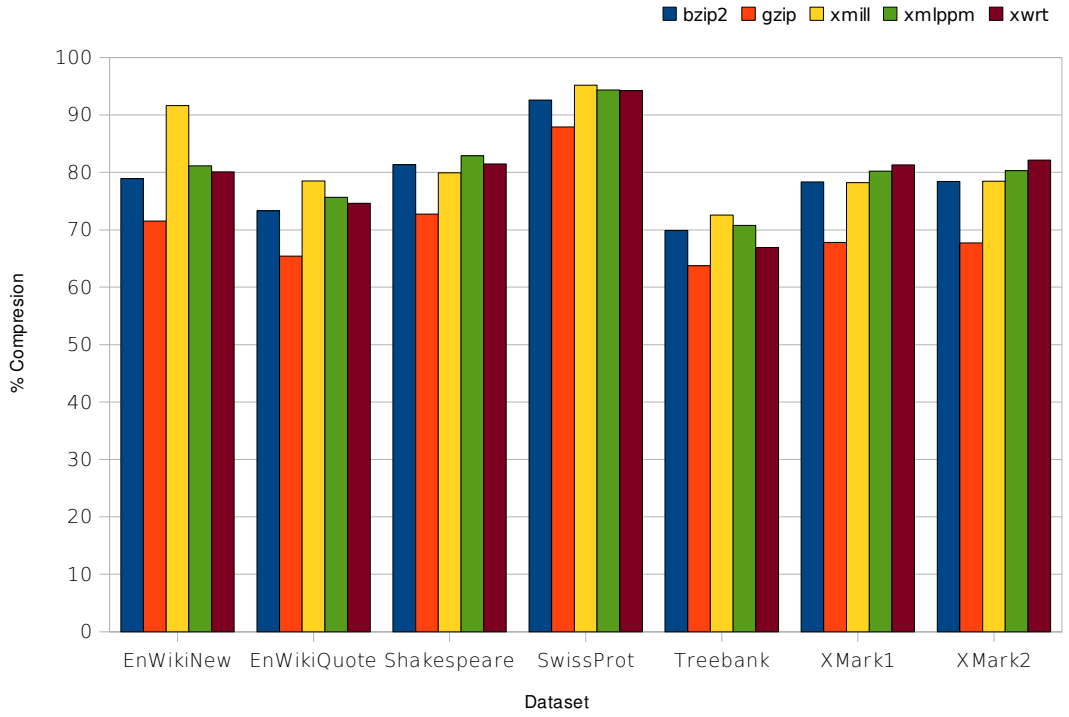

Figura 5.3: Compression Ratio 2 (promedio por dataset)

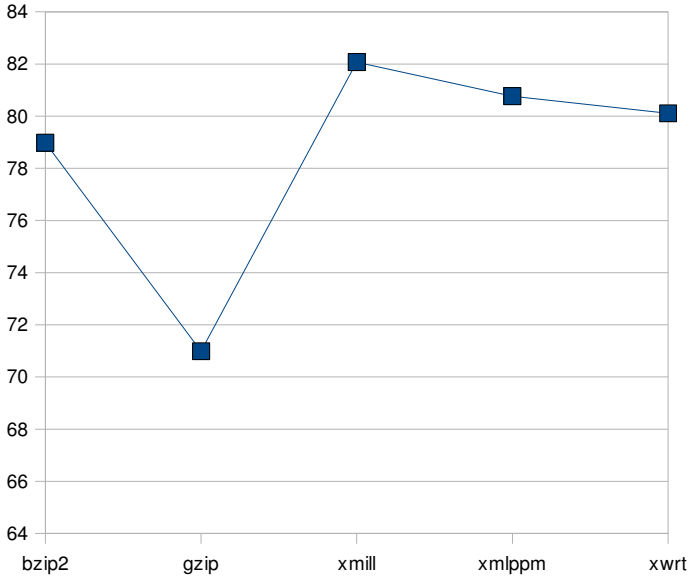

FIGURA 5.4: Compression Ratio 2 (promedio)

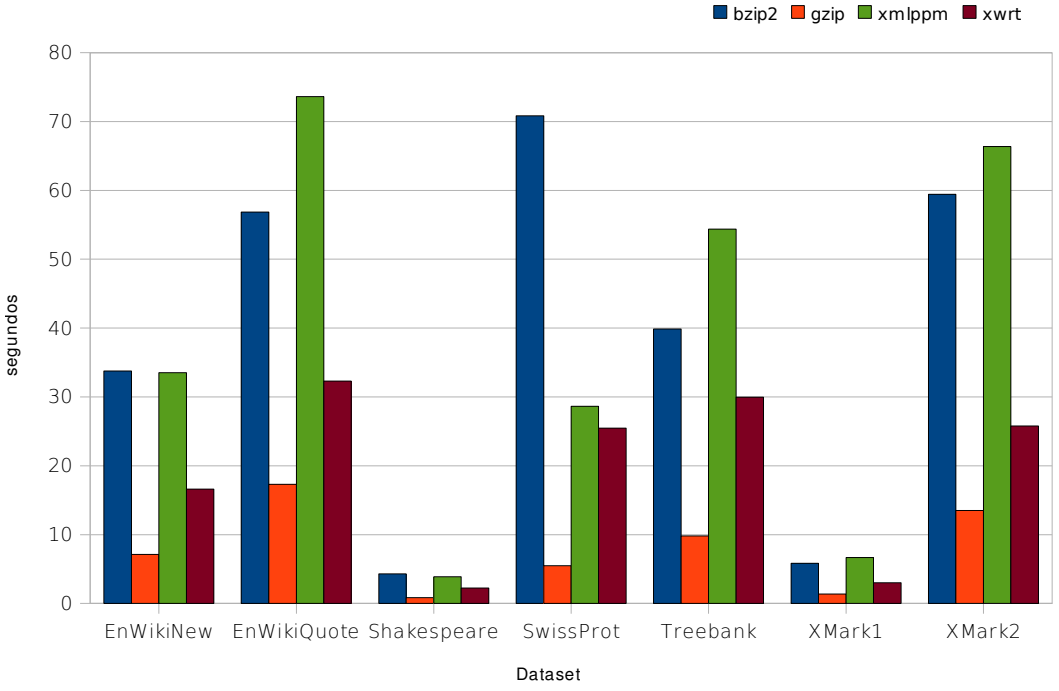

FIGURA 5.5: Tiempos de Compresión (promedio por dataset)

En relación a los resultados e información expuestos, se obtienen las siguientes conclusiones:

Con respecto a la performance, XMill y XWRT (que es una ligera variante del primero) tienen el mejor promedio, ya que logran un buen nivel de compresión comparable en velocidad a gzip. Si bien XMLPPM también tiene un buen CR, los tiempos de compresión son mucho mayores que los de gzip, lo que lo hace poco práctico para un uso frecuente.

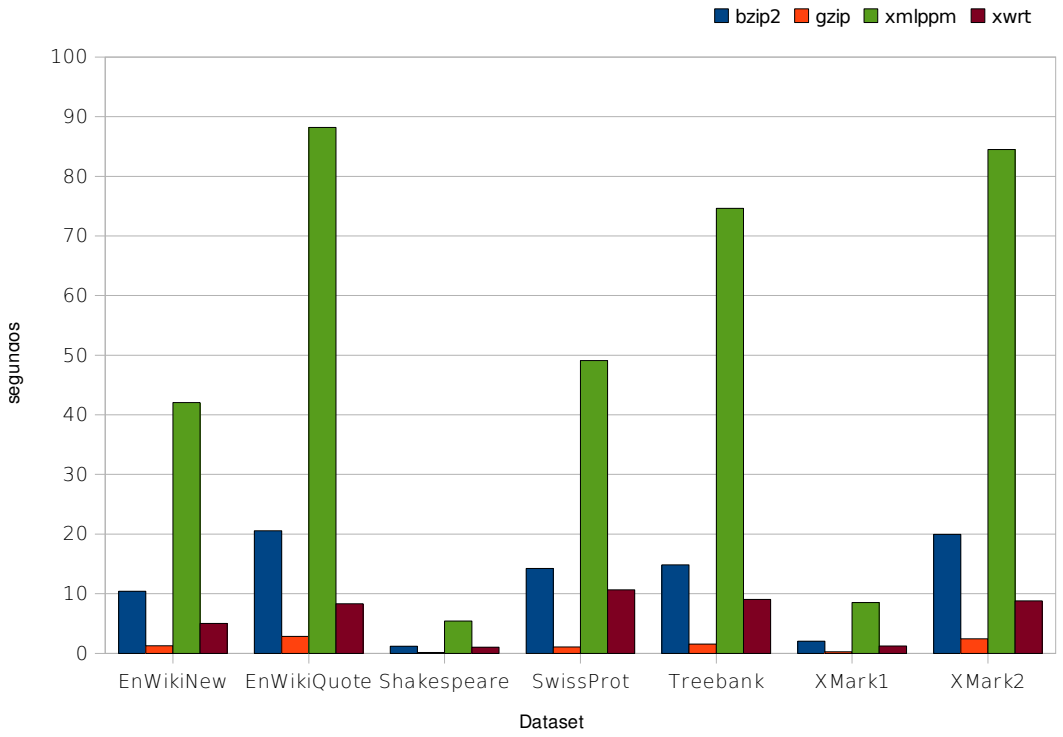

FIGURA 5.6: Tiempos de Descompresión (promedio por dataset)

En base a la bibliografía ([WNC06], [JMC03], [TH02]), y como puede deducirse de las técnicas del capítulo 3, los compresores consultables tienen un nivel de compresión bastante inferior al de gzip y requieren mayores tiempos de procesamiento; sin embargo ofrecen la ventaja de permitir consultas directamente sobre los datos comprimidos.

Si bien no se realizaron medidas del consumo de memoria de cada herramienta, se pueden obtener algunas conclusiones de acuerdo a las técnicas empleadas y la información disponible de cada compresor. Gzip usa un tamaño de buffer fijo de 3MB, independientemente del documento de entrada. XMill como vimos anteriormente [4.1] (también XWRT [4.3]), utiliza una ventana de memoria de tamaño fijo, que una vez completa, el compresor se encarga de escribir a disco (esos datos comprimidos) y luego reanudar el proceso de compresión. Por otro lado, XMLPPM construye cuatro modelos PPM para un documento de entrada, y estos modelos tienen un tamaño fijo, por lo tanto el consumo de memoria también está limitado.

Por el lado de los compresores consultables, XGrind [4.6] (y similarmente, XPress [4.7]) usa la codificación de Huffman para comprimir los datos y entonces solamente necesita memoria para mantener el estados de los modelos de Huffman, y como además estos modelos no varían mucho entre documentos, el consumo de memoria está bastante delimitado.

El principal problema es con aquellos compresores que parsean el documento usando DOM, es decir que necesitan cargar en memoria el documento, requiriendo en este caso un consumo de memoria proporcional al tamaño del documento de entrada, lo que los convierte en una mala opción cuando el documento XML a comprimir es muy grande.

Finalmente, en relación a la performance y alcance de las consultas posibles sobre compresores consultables se pueden aportar algunos datos en función a la información publicada en los respectivos papers dado que, o bien no hay implementaciones disponibles, o no se pudieron evaluar.

De acuerdo a los resultados en [TH02], las consultas en XGrind son de 2 a 3 veces más rápidas que utilizar las consultas nativas sobre el parser de XMill (es decir, descomprimiendo el archivo para hacer la consulta). Según [JMC03], XPress efectúa consultas 2.83 veces más rápido que XGrind, mientras que los resultados experimentales de XQzip [CN04] dan 12.84 veces mejor que XGrind.

Bajo estas condiciones, se puede concluir que en aquellas situaciones en que la compresión de un documento es necesaria (según lo expuesto en [1.3], por ejemplo), un compresor específico representa una buena opción ante los compresores de texto general, sobre todo XMill y XWRT que logran tiempos que se acercan bastante a los de gzip, pero logran un nivel de compresión, en promedio, bastante mejor.

Es claro que con el uso de un compresor no consultable se sacrifican algunas ventajas del documento original, aunque son las mismas que se pierden con un compresor de texto general. En este sentido, si la idea es reducir el tamaño original y soportar frecuentes ciclos de compresión/descompresión de un documento XML, la utilización de XMill o XWRT resulta una alternativa a considerar. XMLPPM, en cambio, no resulta recomendable dado sus altos tiempos de procesamiento (salvo quizás para un almacenamiento de respaldo).

Un caso de aplicación práctica se puede ver en el intercambio de mensajes SOAP sobre una red con ancho de banda limitado. Los mensajes SOAP pueden llegar a ser grandes, y esto puede tener efectos adversos en la performance debido al incremento de tiempo en la transmisión y recepción de los mismos. Este problema se hace más notorio en el caso de los dispositivos móviles con conexiones limitadas. Como por lo general aumentar las capacidades del cliente suele ser más fácil (e incluso menos costoso) que aumentar el ancho de banda, añadir un ciclo de compresión/descompresión de los mensajes SOAP para reducir su tamaño es una solución razonable.

Si el objetivo es acceder a la información en el documento, permitiendo consultas relativamente simples, y el tamaño es una restricción pero no así los tiempos de compresión, entran en juego los compresores consultables. Estos compresores actualmente surgen como una elección a tener en cuenta en los casos en que hay necesidad de comprimir el documento, una única vez (o que sufren cambios cada períodos largos, que implicarían una nueva compresión), y luego acceder a esos datos frecuentemente.

Una situación en la que el uso de un compresor consultable puede ser útil, es el caso en que se cuenta con bases de datos grandes en las que hay cierta información concisa sobre la que se realizan la mayoría de las consultas, y un gran volumen de texto. Tal es el caso, por ejemplo, de un documento XML que cataloga libros, que además de título, autor, ISBN y tags, incluye el texto de los mismos (o un resumen). Entonces se puede conseguir un buen nivel de compresión del documento, pero aún así permitir búsquedas sobre él.

Capítulo 6

Comparación Final

En este último capítulo se presenta un resumen de la información analizada a lo largo del trabajo mediante una tabla comparativa de las características de los distintos compresores.

	XMill	XComp	XWRT	eXalt	XMLPPM
Disponible	Sí	No	Sí	Sí	Sí
Licencia	BSD, GPL	-	GPL	GPL	GPL
Url	`http://sourceforge.net/projects/xmill/`	-	`http://xwrt.sourceforge.net`	`http://exalt.sourceforge.net`	`http://xmlppm.sourceforge.net`
Clasificación	Independiente de esquema. No consultable. Offline.	Independiente de esquema. No consultable. Offline.	Independiente de esquema. No consultable. Offline.	Independiente de esquema. No consultable. Offline.	Independiente de esquema. No consultable. Online.
Compresor general?	gzip, bzip2, PPM	zlib, Huffman	gzip, lzma, PPM	Compresión sintáctica (gramáticas)	PPM
Técnicas	Separación de estructura y datos. Contenedores por elemento. Compresores semánticos.	Separación de estructura y datos. Contenedores por elemento, considerando nivel y tipo de datos. Codificación específica para números y fechas.	Separación de estructura y datos. Contenedores por elemento. Diccionario de palabras frecuentes.	Representación como gramática. Uso de modelos probabilísticos. Emite eventos SAX en descompresión.	Uso de modelos probabilísticos. Idea de Predicción por Coincidencia Parcial (PPM).
Compression Ratio 1	1.43 bits/byte	-	1.59 bits/byte	-	1.54 bits/byte
Compression Ratio 2	82.07 %	-	80.11 %	-	80.76 %
Consumo de memoria	Limitado (usa ventana de tamaño fijo)	Limitado (usa ventana de tamaño fijo)	Limitado (usa ventana de tamaño fijo)	Limitado (aproximadamente constante)	Limitado (modelos de tamaño fijo)
Consultas posibles	No	No	No	No	No

	XGrind	XPress	XQueC	XQzip
Disponible	Sí	No	No	No
Licencia	GPL	-	-	-
Url	`http://xgrind.sourceforge.net/`	-	`http://staff.icar.cnr.it/angela/xquec/`	-
Clasificación	Dependiente de esquema. Consultable. Online.	Independiente de esquema. Consultable. Online.	Independiente de esquema. Consultable. Online.	Independiente de esquema. Consultable. Online.
Compresor general?	Huffman	Huffman, codificadores numéricos, diccionario	Huffman	-
Técnicas	Diccionario de elementos y atributos. Codificadores de Huffman. Transformación homomórfica.	Codificación Aritmética Inversa. Diccionario de elementos y atributos. Transformación homomórfica.	Separación de estructura y datos. Mantiene estructura del árbol del documento. Mantiene listado de caminos. Uso de contenedores	Árbol de estructura indexado. Compresión por bloques.
Compression Ratio 1	-	-	-	-
Compression Ratio 2	-	-	-	-
Consumo de memoria	Limitado (dado por codificadores de Huffman)	Limitado (dado por codificadores de Huffman)	-	-
Consultas posibles	Sí. Expresiones XPath sobre los ejes hijo y atributo. Coincidencia exacta y de prefijo	Sí. Expresiones XPath sobre los ejes hijo, atributo y descendiente. Coincidencia exacta y de prefijo. Búsqueda por rango en datos numéricos.	Sí. Subconjunto de XQuery. Coincidencia exacta y de prefijo.	Sí. Subconjunto de XPath.

Capítulo 7

Conclusiones

- XMill es el primer compresor específico de importancia, aún vigente, y es la herramienta que introduce las primeras innovaciones en materia de compresión de XML, tales como la separación de datos y estructura, la agrupación por contenedores y el uso de compresores semánticos, ideas que toman luego la mayoría de las variantes posteriores.
- En la práctica gran parte de los compresores de XML se basan en algún preprocesamiento sobre la estructura del documento, y ese resultado se pasa por un compresor de propósito general. En consecuencia, los niveles de compresión de muchos de los compresores específicos de XML se relacionan y dependen de los compresores generales que usan por detrás (gzip, bzip2, PPM). Como se puede ver en los resultados de los experimentos ninguno de los compresores específicos se destaca mucho sobre las herramientas de compresión general. Esto podría explicar el hecho de que los compresores específicos no se usen de manera más amplia.
- Para muchas de las herramientas descriptas en la literatura, especialmente los compresores consultables, no se encuentra el código disponible, haciendo difícil evaluar las técnicas expuestas o hacer experiencias repetibles que permitan verificar los números que dicen alcanzar.
- No hay implementaciones sólidas de compresores de XML basados en gramáticas o de compresores consultables. Éstas parecen ser áreas de interés para investigación y desarrollo.
- Otro de los desafíos a futuro es el de equilibrar el nivel de compresión y la velocidad, así como la performance de las consultas y su expresividad, características que en las implementaciones actuales se sacrifican una en pos de la otra.
- La evolución de los compresores consultables permitiría quizás su uso en bases de datos como forma más eficiente para el almacenamiento nativo de datos en formato XML y la posibilidad de efectuar consultas sobre esos datos.
- La usabilidad de las tecnologías de compresión no está del todo clara en el contexto moderno de aplicaciones web. Es útil estudiar la posibilidad de adoptar los desarrollos vistos en

diferentes servicios y su interoperabilidad en arquitecturas web (clientes/servidor, web services, mediadores, dispositivos móviles, etc).

- Muchas de las herramientas propuestas no pasan de su etapa de prototipo o prueba, o dejan de mantenerse en el tiempo, lo cual impide una mayor difusión y la posibilidad de incorporar la compresión específica de XML como una alternativa habitual.

Apéndice A

Código Fuente

En el capítulo 5 se hizo una evaluación de la performance de los distintos compresores de XML. Para efectuar los tests se utilizó el siguiente script Python (desarrollado por el autor de este trabajo):

```
1  import subprocess
2  import os
3  import csv
4
5  # Some settings
6  runs = 2                                        # number of runs
7  dataset_directory = 'dataset'                   # datasets base directory
8  test_directory = 'test'                         # temporal directory for compressed
       files/decompression
9  results_file = 'compression_results.csv'        # csv file where results are written
10
11 # Compressors data
12 compressors = {
13                 #'compressor name': {'cmd': 'path_to_compressor_bin',
14                 #                    'extension': '.compressor_output_extension',
15                 #                    'extra_params': ('param1', 'param2'),
16                 #                    '>out?': True, # (redirect stdout?),
17                 #                    'uncmd': 'path_to_decompressor_bin',
18                 #                    'uncmd_file?': '
       path_to_decompressor_destination_file'
19                 #                  }
20                 'gzip': {'cmd':'gzip',
21                          'extension':'.gz',
22                          'extra_params': ('-c',),
23                          '>out?': True,
24                          'uncmd': 'gunzip',
25                          'uncmd_file?': False,
26                          },
27                 'bzip2': {'cmd':'bzip2',
28                           'extension':'.bz2',
29                           'extra_params': ('-c',),
30                           '>out?': True,
31                           'uncmd': 'bunzip2',
```

```
32                          'uncmd_file?': False,
33                          },
34                 'xwrt': {'cmd':'./xwrt/xwrt_linux',
35                          'extension':'.xwrt',
36                          'extra_params': (),
37                          '>out?': False,
38                          'uncmd': './xwrt/xwrt_linux',
39                          'uncmd_file?': False,
40                          },
41                 'xmlppm': {'cmd':'./xmlppm-0.98.3/src/xmlppm',
42                            'extension':'.xpm',
43                            'extra_params': (),
44                            '>out?': False,
45                            'uncmd': './xmlppm-0.98.3/src/xmlunppm',
46                            'uncmd_file?': True,
47                            },
48                 }
49
50 # datasets info (Name: File)
51 datasets = {'Shakespeare': 'Shakespeare.xml',
52             }
53
54 # Open destination csv file
55 results_fd = open(results_file, 'w')
56 results_writer = csv.writer(results_fd)
57
58 # Write column headers
59 results_writer.writerow(['compressor', 'dataset', 'run', 'compression ratio 1',
60                          'compression ratio 2', 'compression time', '
       decompression time'])
61
62 # i: run
63 for i in range(runs):
64     for comp_id, comp_params in compressors.iteritems():
65         for xml_file_id, xml_file in datasets.iteritems():
66             # input file
67             input_file = os.path.join(dataset_directory, xml_file)
68
69             # output file
70             output_file_name = xml_file + comp_params['extension']
71             output_file = os.path.join(dataset_directory, output_file_name)
72
73             #### Compression Times and Ratio
74
75             # build command
76             cmd = (comp_params['cmd'],) + comp_params['extra_params'] + (
       input_file,)
77
78             # check if we need to redirect the output
79             stdout = None
80             if comp_params['>out?']:
81                 stdout = open(output_file, 'w')
82
83             # run the compressor
84             p = subprocess.Popen(cmd, stdout=stdout)
```

```
85
86              # if output was redirected, close the handle
87              if stdout:
88                  stdout.close()
89
90              # get return values and compression processing times
91              ret = os.wait4(p.pid, 0)
92              compression_time = ret[2][0]
93
94              assert(os.access(output_file, os.F_OK))
95
96              # get original and compressed files info (compression ratio)
97              original_file_info = os.stat(input_file)
98              compressed_file_info = os.stat(output_file)
99              compression_ratio1 = compressed_file_info.st_size * 8.0 /
    original_file_info.st_size
100             compression_ratio2 = (1 - 1.0 * compressed_file_info.st_size /
    original_file_info.st_size) * 100
101
102             #print comp_id, xml_file_id, time, original_file_info.st_size,
    compressed_file_info.st_size
103
104             #### Decompression Times
105
106             # rename output file
107             temp_file = os.path.join(dataset_directory, test_directory,
    output_file_name)
108             os.renames(output_file, temp_file)
109
110             # decompressed file
111             decompressed_file = os.path.join(dataset_directory, test_directory,
    xml_file)
112
113             # build command
114             cmd = (comp_params['uncmd'], temp_file)
115
116             # some compressors need to specify the destination file
117             if comp_params['uncmd_file?']:
118                 cmd = cmd + (decompressed_file,)
119
120             # run the decompressor
121             p = subprocess.Popen(cmd)
122
123             # get return values and compression processing times
124             ret = os.wait4(p.pid, 0)
125             decompression_time = ret[2][0]
126
127             assert(os.access(decompressed_file, os.F_OK))
128
129             #print comp_id, time
130
131             # clean test dir
132             os.remove(decompressed_file)
133             if os.access(temp_file, os.F_OK):
134                 os.remove(temp_file)
```

```
135
136             # add results row to csv file
137             row = [comp_id, xml_file_id, i+1, compression_ratio1,
    compression_ratio2, compression_time, decompression_time]
138             results_writer.writerow(row)
139
140 # close csv file
141 results_fd.close()
```

Listing A.1: Script Python

Bibliografía

[CAB07] Leemon C. Baird III Dursun A. Bulutoglu Chris Augeri, Barry E. Mullins and Rusty O. Baldwin. An analysis of xml compression efficiency. *Proceedings of the 2007 Workshop on Experimental Computer Science*, 2007.

[CN04] J. Cheng and W.Ñg. Xqzip: Querying compressed xml using structural indexing. *Proceedings of the Ninth International Conference on Extending Database Technology*, 2004.

[JMC03] M. Park J. Min and C. Chung. Xpress: A queriable compression for xml data. *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 2003.

[Li03] W. Li. Xcomp: An xml compression tool. Master's thesis, University of Waterloo, 2003.

[LS00] H. Liefke and D. Suciu. Xmill: An efficient compressor for xml data. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000.

[PSS08] Sz. Grabowski P. Skibiński and J. Swacha. Effective asymmetric xml compression. *Software: Practice and Experience*, 2008.

[Sak08] Sherif Sakr. An experimental investigation of xml compression tools. *The Computing Research Repository*, 2008.

[TH02] P. Tolani and J. Haritsa. Xgrind: A query-friendly xml compressor. *Proceedings of the 2002 International Conference on Data Engineering*, 2002.

[Tom03] Vojtech Toman. Compression of xml data (exalt). Master's thesis, Charles University, Prague, 2003.

[WNC06] Lam Wai Yeung WilfredÑg and James Cheng. Comparative analysis of xml compression technologies. *World Wide Web 9*, 2006.