Externe Sortierung von XML-Dokumenten

Stefanie Scherzinger[1]
scherzin@fmi.uni-passau.de

1 Einleitung

Im Umgang mit XML-Dokumenten ist es für Archivierung und Versionskontrolle erforderlich, Dokumente unterschiedlichen Entwicklungsstands miteinander zu vergleichen oder zu mergen. So ist im Archivierungsverfahren von [3] ein effizientes Sortierverfahren auf dem Sekundärspeicher Voraussetzung für die Skalierbarkeit.

Ein klassischer Merge-Join, wie er für relationale Datenbanken von großer Bedeutung ist, setzt dabei eine Sortierung der Join-Partner voraus. Diese Arbeit beschäftigt sich mit den Eigenschaften, die ein entsprechendes Sortierverfahren für XML gewährleisten soll und stellt eine Datenstruktur zur externen Sortierung vor.

2 Bestehende Arbeiten

LTXML [6] und das *XMLToolkit* [2] stellen Sammlungen Unix-ähnlicher Werkzeuge für den Umgang mit XML dar. Die Befehlssyntax von *sgsort* aus *LTXML* lässt keine rekursiven Sortierungsanweisungen zu. *xsort* aus dem *XMLToolkit* bietet eine schnelle externe Sortierung von XML-Dokumenten und Streams. Es kann auf unterschiedlichen Ebenen im Baum arbeiten, jedoch nicht rekursiv. Zudem sind Daten- und Hierarchieverluste möglich: Alle Pfade und Knoten, die nicht explizit in XPath-Ausdrücken spezifiziert werden, gehen verloren.

Eine Sortierung als Vorstufe zum Merge-Join erfordert ein rekursives Verfahren ohne Daten- oder Hierarchieverluste. Insbesondere stellt eine Sortierung eine Permutation dar, die eine Erhaltung der Daten impliziert. Aus dieser Motivation heraus entstand folgender Vorschlag eines Verfahrens zur externen Sortierung.

3 Eigener Beitrag zum Sortieren von XML

3.1 Spezifikation von Context, Target und Key

Eine Sortierung wird durch Angabe von XPath-Ausdrücken[2] für die Rollen *Context*, *Target* und *Key* spezifiziert. Diese Terminologie lehnt sich an die in [2] an, die Definitionen der Begriffe unterscheiden sich in Details.

Ein *Context* ist ein Pfad von der Wurzel bis zu einem Knoten im Baum. Unter seinen direkten Kindern sind die zu sortierenden Knoten, die *Targets*, anzugeben. Jedem *Target* sind *Keys* zugeordnet. *Keys* sind direkte und indirekte Kinder des *Targets*. Bei mehr als einem *Key* wird hierarchisch sortiert.

Für eine rekursive Sortierung werden *Context*, *Target* und *Key* geschachtelt deklariert. Im folgenden Beispiel einer Bücherdatenbank im XML-Format werden alle Bücher unter `<lib>` nach ihrem Titel sortiert, d. h. `context=/lib`, `target=book`, `key=title`. Innerhalb der Bücher sind alle Autoren nach Namen zu sortieren, d. h. `context=book`, `target=author`, `key=name`.

Wie in Abbildung 1*a*) dargestellt, werden dadurch Knotenmengen im XML-Baum spezifizert. Die Sortierung aller Bücher bewirkt die Permutation der Teilbäume mit Wurzel `<book>`.

[1]Dieses Projekt entstand im Rahmen eines Auslandsaufenthaltes an der University of Edinburgh unter der Betreuung von Prof. Buneman und Dr. Christoph Koch und wurde bei Prof. Kemper am Lehrstuhl für Dialogorientierte Systeme an der Universität Passau fortgesetzt.

[2]Wo XPath Ausdrücke Wildcards beinhalten, wird nach *First Match Fits* vorgegangen.

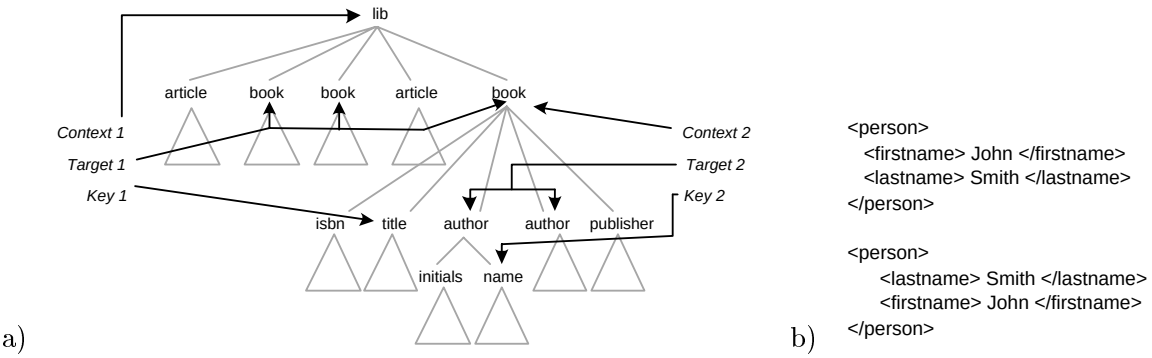

Abbildung 1: a) Rollenverteilung für Knoten b) nicht-äquivalente XML-Fragmente nach [1]

Teilbäume mit Wurzel `<article>` behalten ihre Position bei. Analoges gilt für `<author>`-Knoten unter einem `<book>`-Knoten. In Anlehnung an Merge-Sort wurde hierfür ein Algorithmus mit vergleichbarer Komplexität entwickelt.

3.2 Bildung von Schlüsselwerten

Als Sortierschlüssel dienen Werte von Knoten. Den Wert eines Knotens `<n>` bildet man durch Konkatenation der Strings im Teilbaum mit Wurzel `<n>`. Da XML-Dokumente geordnet sind, ergeben sich im Beispiel aus Abb. 1*b*) zwei verschiedene Werte für `<person>`: „John Smith“ und „Smith John“. Liegt eine Sortieranweisung für den Teilbaum unter einem Schlüsselknoten vor, wird diese ausgeführt, bevor der Schlüsselwert gebildet wird. So kann gewährleistet werden, dass Schlüsselwerte deterministisch gebildet werden.

3.3 Sortierverfahren und Datenstruktur

Das Ausgangsdokument wird in einem Durchlauf vom SAX-Parser gelesen. Im Hauptspeicher entsteht eine Baumstruktur (s. Abb. 2), die die XML-Daten fragmentiert. Die Kohäsion zwischen zu sortierenden und nicht zu sortierenden Daten sowie zwischen den Ebenen bleibt gewahrt. Wenn das Dokument eingelesen ist, ist die Datenstruktur komplett erstellt und durch das bottom-up Verfahren sortiert. Ein Durchlauf des Baumes in Tiefensuche und die Konkatenation aller seiner Blätter liefern das sortierte Dokument.

Falls während der Erstellung der Datenstruktur die Speicher-Allokation überschritten wird, werden bereits sortierte Teilbäume in Dateien ausgelagert und beim endgültigen Durchlauf von der Platte geholt. So kann das Verfahren auch für die externe Sortierung von Dokumenten eingesetzt werden. Es wird dabei angenommen, dass mindestens die Datenstruktur und die zur Sortierung gerade benötigten Schlüssel im Hauptspeicher gehalten werden können. Grundlage für die hier vorgestellte Technik bilden ein *XPath-Prozessor* [5] und ein *Tokenized SAX-Parser* (TSAX) [2], wie sie im XMLToolkit vorgestellt werden. Das Verfahren wurde in C++ unter Verwendung von Bibliotheken des XMLToolkit [7] implementiert und wird im folgenden anhand des Beispiels aus 3.1 vorgestellt.

Die rekursiv deklarierten Sortieranweisungen werden im XPath-Prozessor, einem „lazy“ Automaten zur Erkennung von XPath-Ausdrücken, registriert. Durch Kopplung mit einem Tokenized SAX-Parser werden beim Lesen des Dokuments sowohl die normalen SAX-Events (`start`- bzw. `end_dokument`, `start`/`end_tag`) als auch Events für registrierte XPath-Ausdrücke initiiert. Eine Umsetzungstabelle ordnet jedem XPath-Event eine Menge von Rollen zu.

Die Baumstruktur entsteht in Reaktion auf die Events des TSAX-Parsers: Mit Beginn eines XPath-Events wird ein Kindknoten in den Baum eingefügt. Auf das Ende eines XPath-Events wird je nach Rolle des Knotens reagiert. Besitzt ein Knoten mehrere Rollen werden alle entsprechenden Aktionen ausgeführt. Hat der eben abgearbeitete Knoten die Rolle eines *Keys*, so stellt

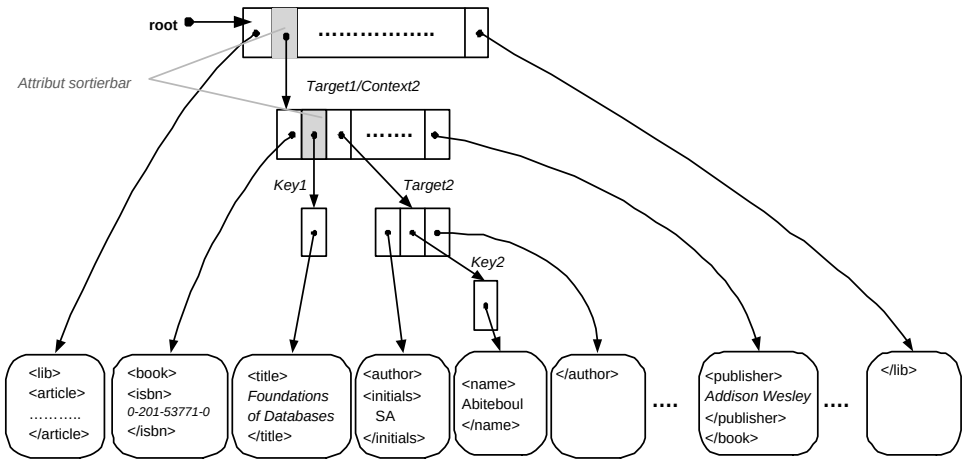

Abbildung 2: *Aufbau der Baumstruktur*

sein Wert einen Schlüssel dar. Dieser Schlüsselwertes wird in einen zentralen Puffer kopiert. Mit Ende eines *Targets* erhält der aktuelle Knoten das Attribut `zu_sortieren`. Seine Schlüsselwerte liegen gesammelt im zentralen Puffer bereit. Bei Abschluss eines *Contexts* wird der aktuelle Knoten sortiert. Dazu werden alle Einträge mit dem Attribut `zu_sortieren` permutiert. Unabhängig von der speziellen Rolle wird im Anschluß der aktuelle Knoten verlassen. Sämtliche normalen SAX-Events werden in den Blättern des Baumes als XML-Fragmente gepuffert. Abbildung 2 zeigt die Datenstruktur in Ausschnitten.

4 Zusammenfassung und Ausblick

Falls nur Teilbereiche eines Dokuments sortiert werden sollen, stellt diese Vorgehensweise ein effizientes Verfahren zur rekursiven Sortierung dar. Soll ein großes Dokument komplett auf allen Ebenen sortiert werden, kann es zur temporären Auslagerung vieler kleiner Dateien auf den Sekundärspeicher kommen. Nachdem diese in neuer Reihenfolge eingelesen werden, ergeben sich ungünstige Verzögerungen.

Daher ist es angebracht, intelligente Auslagerungsstrategien zu entwickeln. So könnte bei Auslagerung nach dem Grundsatz *Large Blobs First* vorgegangen werden. Hierbei werden erst die großen Teilbäume hinausgeschrieben in der Hoffnung, dass wieder genügend Speicher für die restliche Sortierung frei wird. Vor allem die Weiterführung der Datenstruktur in ein Merge-Join Verfahren für XML-Dokumente ist von Interesse.

Literatur

[1] S. Abiteboul, P. Buneman, D. Suciu. Data on the Web - From Relations to Semistructured Data and XML. *Morgan Kaufmann Publishers*, 2000.

[2] I. Avila-Campillo, T. Green, A. Gupta, M. Onizuka, D. Raven, D. Suciu. XMLTK: An XML Toolkit for Scalable XML Stream Processing. *PLANX*, 2002.

[3] P. Buneman, S. Khanna, K. Tajima, W. Tan. Archiving Scientific Data. *SIGMOD*, 2002.

[4] P. Buneman, S. Davidson, W. Fan, C. Hara, W. Tan. Keys for XML. *WWW10*, 2001.

[5] T. Green, G. Miklau, M. Onizuka, D. Suciu. Processing XML Streams with Deterministic Automata. *Proc. ICDT*, 2003.

[6] Edinburgh Language Technology Group. LT XML version 1.2.5. `http://www.ltg.ed.ac.uk/software/xml/`

[7] XMLTK: An XML Toolkit for Lightweight XML Stream Processing. `http://xmltk.sourceforge.net/`