Migration nicht-nativer XML-Strukturen in den nativen XML-Datentyp am Beispiel von DB2 for z/OS V9.1

Christoph Koch

Friedrich-Schiller-Universität Jena
Lehrstuhl für Datenbanken und Informationssysteme
Ernst-Abbe-Platz 2
07743 Jena
Christoph.Koch@uni-jena.de

DATEV eG
Datenbanken
Paumgartnerstr. 6-14
90429 Nürnberg
Christoph.Koch@datev.de

KURZFASSUNG

Mit der Einführung der pureXML-Technologie [4] bietet das von IBM entwickelte Datenbankmanagementsystem (DBMS) DB2 for z/OS die Möglichkeit, XML-Daten nativ zu speichern und XML-spezifisch zu verarbeiten. Die native Art der XML-Ablage verspricht gegenüber der nicht-nativen XML-Speicherung zahlreiche Vorzüge, die es sinnvoll werden lassen, im Unternehmen vorhandene nicht-native XML-Daten in die native Struktur zu überführen. Der vorliegende Beitrag setzt genau an dieser Migrationsthematik auf und zielt darauf ab, sowohl die einzelnen für eine solche Migration notwendigen Schritte aufzuzeigen, als auch deren Performance in Relation zum Gesamtaufwand der Überführung nicht-nativer XML-Dokumente in die native XML-Speicherform zu evaluieren.

Die Ergebnisse dieses Beitrags basieren auf der Grundlage der Arbeiten [1] und [2], die im Rahmen einer umfassenden Untersuchung der XML-Unterstützung von DB2 for z/OS V9.1 seit 2010 in der DATEV eG angefertigt wurden und richten sich daher insbesondere an den dortigen Anforderungen aus.

Kategorien und Themenbeschreibung

Database Performance and Benchmarks

Allgemeine Bestimmungen

Management, Measurement, Performance, Design, Languages

Schlüsselwörter

XML, nicht-nativ, nativ, Migration, Methodik, Performance

1. EINLEITUNG

Seit die Extensible Markup Language (XML) im Jahr 1998 vom World Wide Web Consortium (W3C) verabschiedet wurde, entwickelte sie sich fortlaufend weiter zum De-facto-Standard für den Austausch von (semi-)strukturierten Daten. Damit einher ging auch der verstärkte Einzug von XML in den Datenbanksektor [6], sodass sich sowohl dedizierte XML-DBMS herausbildeten, als auch herkömmliche relationale Systeme um XML-Strukturen erweitert wurden.

Die einfachste und älteste Form, XML-Daten in relationalen DBMSen zu speichern, ist die nicht-native XML-Speicherung, die die datenbankseitige Ablage von XML-Daten als Binär- oder Character-Strings vorsieht [7]. Demgegenüber existieren seit jüngerer Zeit auch native XML-Speichermöglichkeiten, wie beispielsweise die hier betrachtete in DB2 implementierte pureXML-Technologie und der damit verbundene native XML-Datentyp.

Da die native XML-Speicherung verglichen mit den verschiedenen nicht-nativen Ablageformen diverse Vorteile hinsichtlich Performance, Flexibilität und Kosteneffizienz verspricht [4], besteht auf langfristige Sicht das Ziel darin, „alte“ nicht-native Datenbestände in die „neue“ native Form der XML-Speicherung zu überführen. Die dazu notwendige Migration charakterisiert sich im Kontext von DB2 for z/OS durch den in den folgenden Abschnitten beschriebenen Ausgangs- und Zielzustand.

1.1 Ausgangszustand

Den Ausgangszustand der hier betrachteten Migrationsthematik bilden nicht-native XML-Dokumente, die im Fall der DATEV eG vorrangig in Form des VARCHAR- oder CLOB-Datentyps als Character-String und noch spezieller in bestimmten Kontexten über mehrere Tupel verteilt gespeichert sind. Solche Dokumente sind in unserem Kontext zum Beispiel von einer OCR-Software ins XML-Format digitalisierte Belege. Auf die genannten Besonderheiten der nicht-nativen Speicherformen wird nun konkreter eingegangen.

24th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012-01.06.2012, Lübbenau, Germany.

„Nicht-nativ“ bedeutet der Wortherkunft nach „nicht natürlich“. Bezogen auf die nicht-native oder auch XML-enabled genannte XML-Speicherung resultiert daher ein Unterschied zwischen Speicher- und Verarbeitungsformat, sodass auf diese Weise gespeicherte XML-Dokumente ein Mapping von dem vorliegenden (relationalen) Speichermodell hin zu dem XML-Datenmodell benötigen [5]. Beispiele für nicht-native XML-Speicherformen sind die Ablage von XML-Dokumenten als Binär- oder Character-String (siehe *Abbildung 1*) oder auch das Konzept der XML-Dekomposition. Letzteres basiert auf der Zerlegung von XML-Dokumenten in relationale Strukturen. DB2 stellt hierzu die XMLTABLE-Funktion [11] bereit.

Die weiteren Ausführungen dieses Beitrags konzentrieren sich aber wegen ihrer höheren Relevanz für die DATEV eG ausschließlich auf die nicht-native XML-Speicherform als Character-String und betrachten dabei speziell die Verwendung des CLOB-Datentyps. In diesem Fall wird für die betrachtete Migration als Mapping die zentrale Funktionalität des XML-Parsens benötigt. Dabei werden nicht-native XML-Daten in die native Speicherform transformiert. Nähere Details dazu folgen in Abschnitt 2.4.

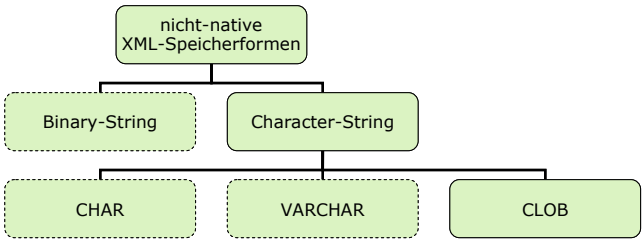

Abbildung 1: nicht-native XML-Speicherformen (Auswahl)

In der DATEV eG werden XML-Daten zum aktuellen Zeitpunkt nicht-nativ in der EBCDIC-Codepage gespeichert. Dabei handelt es sich um einen seit langem etablierten Single-Byte-Zeichensatz, der zur Repräsentation von Character-Daten genutzt wird und nur einen verglichen mit heutigen Standards wie Unicode geringen Zeichenvorrat abdeckt.

Für die hier betrachtete nicht-native XML-Speicherform des CLOB-Datentyps gelten diverse Einschränkungen hinsichtlich seiner Länge. Zwar kann ein CLOB-Dokument DB2-seitig prinzipiell eine maximale Länge von 2 GB besitzen. Diese lässt sich jedoch durch die DB2-Subsystem-Parameter LOBVALA [8] und LOBVALS [9] weiter einschränken, sodass per Default zur Ressourceneinsparung bei der (Haupt-)Speicheradressierung maximal 10 MB große LOB-Dokumente in DB2 verarbeitet werden können. In der DATEV eG wird von diesem Standardwert geringfügig nach oben abgewichen (aktuell 30 MB). Somit fällt die tatsächlich mögliche Maximallänge eines CLOB-Dokuments in unserem Kontext wesentlich kleiner als die genannten 2 GB aus.

Nicht zuletzt infolge dieser praktizierten Einschränkungen hat sich zur DB2-seitigen Speicherung großer XML-Dokumente in der DATEV eG das nachfolgend beschriebene Verfahren etabliert. Derartige Dokumente werden anwendungsseitig, wie in *Abbildung 2* beispielhaft dargestellt, nötigenfalls in kleine, in der Regel nicht wohlgeformte Teile „zerschnitten“ und diese über mehrere (CLOB-)Tupel hinweg datenbankseitig abgelegt. Eine ähnliche Diskussion wird in [1] und [2] auch für die nicht-native XML-Speicherung als VARCHAR geführt.

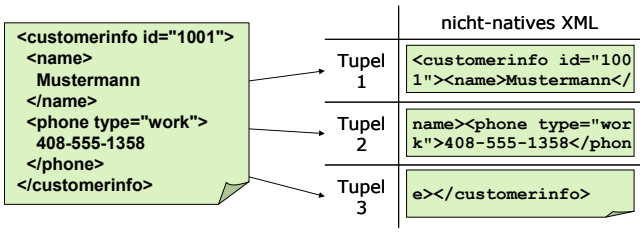

Abbildung 2: Verteilung über mehrere Tupel

1.2 Zielzustand

Entgegen dem zuvor beschriebenen Ausgangszustand erfordert der native Zielzustand der in diesem Beitrag betrachteten XML-Migration keine Variationen. Dies begründet sich durch die eine in DB2 implementierte, optimierte und sich implizit selbst verwaltende native XML-Speicherform, die auf speziell angepassten, aus der relationalen Welt bereits bekannten Strukturen basiert (siehe *Abbildung 3*). Hierzu zählen, ähnlich zur LOB-Speicherung, in XML-Tablespaces befindliche XML-Auxiliary-Tabellen, die über DocIDs und NodeIDs mit den Ausgangstabellen verknüpft sind. Die gesamten XML-Strukturen sind aber so gestaltet, dass sie aufgrund ihrer impliziten Verwaltung durch DB2 dem Nutzer gegenüber transparent bleiben und ihm somit nur als „normale“ Spalte vom Datentyp XML sichtbar werden. Dadurch hat er allerdings auf der anderen Seite unter anderem keinen auch Einfluss auf den verwendeten Zeichensatz, sodass der native XML-Datentyp in DB2 stets die Unicode-Codepage UTF-8 nutzt [10].

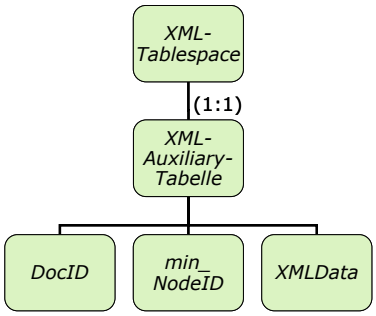

Abbildung 3: implizite XML-Verwaltungsobjekte (nach [10])

2. MIGRATIONSSCHRITTE

Ziel einer Datenmigration ist es, Informationen aus bestehenden Ausgangsstrukturen in spezielle Zielstrukturen zu transformieren. Die dazu notwendigen Teilschritte sind daher die direkte Konsequenz der Differenzen beider Strukturen. Übertragen auf die hier betrachtete Migration leitet sich auf diese Weise die in *Abbildung 4* dargestellte Folge von Migrationsschritten ab, auf die in den kommenden Abschnitten näher eingegangen wird.

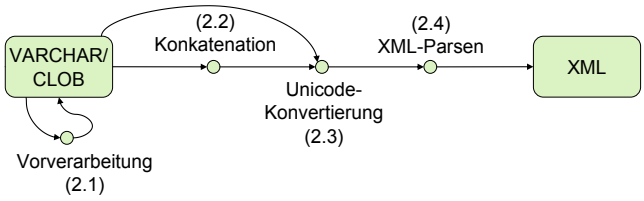

Abbildung 4: Ablauf der Migration

2.1 Vorverarbeitung

Der Schritt der Vorverarbeitung nimmt eine besondere Rolle in der Abfolge der Migration ein, da seine Bedeutung zum einen nicht direkt aus der geschilderten Zustandsdifferenz deutlich wird. Zum anderen lässt sich eine Vorverarbeitung aber auch von der eigentlichen Migrationsfolge (siehe *Abbildung 4*) losgelöst durchführen. Dies erklärt sich durch den funktionalen Zweck des dabei stattfindenden Vorgangs, der im Wesentlichen darin besteht, die Ausgangsdaten zu bereinigen.

Ein derartiger Bereinigungsvorgang ist in unserem Szenario nötig, da aufgrund des beschränkten Zeichenvorrats der EBCDIC-Codepage bestimmte Zeichen nicht dargestellt und infolgedessen als Substitution-Character (HEX: 3F) substituiert werden. Dieser Substitution-Character ist jedoch ein ungültiges Zeichen und wird vom XML-Parser in DB2 nicht akzeptiert. Damit nicht-native XML-Dokumente mit enthaltenen ungültigen Zeichen aber dennoch migriert werden können, ist es notwendig, durch eine Vorverarbeitung diese ungültigen Zeichen durch gültige zu ersetzen. Ein solcher Ersetzungsvorgang kann bereits auf dem Ausgangsdatenbestand durchgeführt werden und ist somit für den direkten Migrationsablauf unkritisch. Eine genauere Untersuchung der Performance dieses Vorverarbeitungsschritts erfolgt daher in Abschnitt 3.2 nicht.

2.2 Konkatenation

Während beliebig große XML-Dokumente nicht-nativ durch das in *Abbildung 2* dargestellte Aufteilen DB2-seitig abgelegt werden, gilt diese Einschränkung für die XML-Speicherung unter Verwendung des nativen XML-Datentyps nicht. Zwar werden native XML-Dokumente intern auch auf verschiedene Tupel verteilt gespeichert, dies aber erfolgt implizit knotenabhängig und gestaltet sich dem Nutzer gegenüber somit vollständig transparent [1]. Für die betrachtete Migration resultiert daher folgende Konsequenz: „zerteilte“ nicht-native XML-Dokumente müssen zur Überführung in den nativen XML-Datentyp wieder konkateniert werden.

Um diesen Vorgang der Konkatenation möglichst performant zu gestalten, sind in [1] und [2] die datenbankseitigen Vorgehensweisen Konkatenation per Stored Procedure und Konkatenation per Recursive SQL untersucht worden. Dabei hat sich erstere klar als die effizientere Variante herausgestellt, wobei sich bereits mit Blick auf Abschnitt 3.2 vorwegnehmen lässt, dass auch diese Form der Konkatenation einen erheblichen Anteil am Gesamtaufwand der Migration einnimmt. Wesentlich effizienter als die hier betrachteten Verfahren der expliziten Konkatenation wäre die direkte Unterstützung einer iterativen Eingabe von nicht-nativen Dokumentteilen durch den XML-Parser. Hierzu stellt DB2 stellt allerdings keine Mechanismen bereit. Insgesamt sei aber nochmals darauf hingewiesen, dass die in *Abbildung 2* gezeigte Zerteilung und damit auch die Konkatenation eher eine Art Sonderform darstellt, die nur in verhältnismäßig wenigen Fällen (sehr große XML-Dokumente) anzutreffen ist.

2.3 Unicode-Konvertierung

Zum aktuellen Zeitpunkt überwiegt in unserer Arbeitsumgebung in DB2 deutlich der Anteil von EBCDIC- gegenüber Unicode-Daten. Demzufolge sind auch nicht-native XML-Dokumente wie in Abschnitt 1.1 beschrieben in der EBCDIC-Codepage codiert abgelegt. Da aber für den nativen XML-Datentyp in DB2 strikt die Unicode-Codepage UTF-8 vorgegeben ist, müssen die ursprünglichen EBCDIC-Inhalte im Rahmen einer Migration in den Unicode-Zeichensatz überführt werden. Hierzu dient der Migrationsschritt der Unicode-Konvertierung.

2.4 XML-Parsen

Der wesentlichste Teil bei der Überführung von nicht-nativen XML-Dokumenten in den nativen XML-Datentyp besteht in dem Schritt des XML-Parsens. Dieser dient dazu, die textual (oder binär) repräsentierten Ausgangsdokumente auf ihre Gültigkeit zu überprüfen und in die hierarchische native XML-Struktur zu überführen. Nähere Details zu dieser speziellen (nativen) Speicherform sind bereits in Abschnitt 2.2 angedeutet worden, sollen hier aber nicht weiter vertieft werden. Eine ausführliche Darstellung dazu liefert [1].

Aus theoretischer Sicht lässt sich hinsichtlich der erwähnten Gültigkeitsprüfung nicht ausschließen, dass es unter bestimmten Bedingungen – beispielsweise bei dem Vorhandensein ungültiger Zeichen im Ausgangsdokument (siehe Abschnitt 2.1) – beim XML-Parsen zu Fehlersituationen kommt. Für den speziellen, hier betrachteten Kontext der Migration gilt dies allerdings nur bedingt. Zum einen werden aktuell sämtliche XML-Dokumente in der DATEV eG bereits vor ihrem Einfügen in DB2 anwendungsseitig geparst und zum anderen dient bereits der in Abschnitt 2.1 angesprochene Migrationsschritt dazu, verbleibende Fehler in den Ausgangsdaten zu bereinigen. Sollten dennoch Probleme während der Migration auftreten, sieht das in [1] diskutierte Konzept zur Migration auch für diesen Fall gezielte Fehlerbehandlungsmaßnahmen vor. Nähere Details dazu sollen hier nicht angefügt werden, sodass für die weitere Abhandlung die Korrektheit der Ausgangsdaten angenommen wird.

DB2 nutzt zum XML-Parsen die gleichnamige XMLPARSE-Funktion [11], die neben einer spezifizierbaren Option zur unterschiedlichen WHITESPACE-Handhabung (siehe *Abbildung 5*) auch zur XML-Schemavalidierung herangezogen werden kann. In der Nachfolgeversion 10 des hier betrachteten DB2 for z/OS V9.1 existieren zusätzlich auch andere Möglichkeiten, XML-Dokumente gegen ein XML-Schema zu validieren. Weitere Informationen dazu und zur XML-Schemavalidierung allgemein finden sich in [2]. Die hier diskutierte XMLPARSE-Funktion nutzt intern die z/OS-XML-System-Services [12] und ermöglicht auf diese Weise das effiziente XML-Parsen nicht-nativer XML-Dokumente.

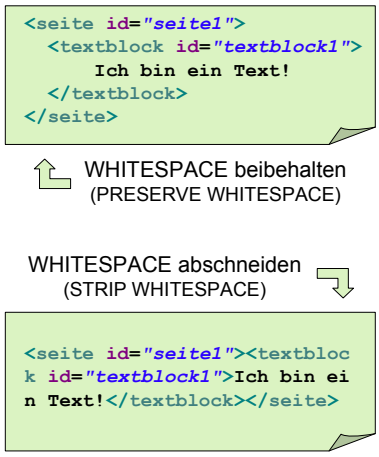

Abbildung 5: WHITESPACE-Handhabung

3. PERFORMANCE

Die Ermittlung von Performance-Kennzahlen zu der in diesem Beitrag betrachteten Migration basiert auf einer Differenzbildung der Kosten verschiedener Folgen einzelner INSERT-Statements von CLOB-Daten bezogen auf das in Abschnitt 3.1 beschriebene Vergleichskriterium. Zur Interpretation der dabei gewonnenen Resultate ist es wichtig, einen groben Überblick über die aktuell in der DATEV eG eingesetzte DB2-Infrastruktur geben. Hierzu seien die folgenden Daten angeführt.

- zSeries = z196 (neueste Generation)
- Betriebssystemversion = z/OS 1.12 [IBM10b]
- Storage-System = Hitachi Data Systems (HDS) – Virtual Storage Platform (VSP)

Die anschließenden Ausführungen beschäftigen sich mit der Darstellung und Analyse der unterschiedlichen, teils unerwarteten Performance-Resultate. Diese finden sich in ausgeweitetem Umfang in [2] wieder und werden dort durch zusätzliche, für diesen Beitrag aber irrelevante Informationen zur Ergebnisdeutung, zum Hintergrund der Messdatenerhebung sowie zur Betrachtung weiterer Kriterien ergänzt.

Generell liegt bei der fortan beschriebenen Erhebung von Performance-Messdaten die Zielstellung darin begründet, den gesamten Migrationsablauf innerhalb von DB2 durchzuführen. Dies schränkt zwar die Möglichkeiten zur Überführung der XML-Daten auf die Verwendung von DB2-Bordmitteln ein – demgegenüber „mächtigere“ Anwendungsbibliotheken werden somit nicht verwendet –, bietet aber die größtmögliche Effizienz, da nur auf diese Weise zusätzlicher Kommunikationsaufwand der zu migrierenden XML-Daten zwischen Anwendung und DB2 eingespart werden kann.

3.1 Vergleichskriterium – Größe und Anzahl

Während [2] die Performance der Migrationsschritte hinsichtlich drei verschiedener Vergleichskriterien analysiert, soll für die Ausführungen dieses Beitrags die Betrachtung des Kriteriums der Größe und Anzahl genügen. Dieses zielt neben einer Gegenüberstellung der Kosten der einzelnen Migrationsschritte auch darauf ab, den Einfluss der Größe der zu migrierenden XML-Dokumente zu untersuchen. Dazu wird ein konstantes Gesamtdatenvolumen vorausgesetzt, das sich durch eine entsprechende Dokumentanzahl reguliert und für die in Abschnitt 3.2 angefügten Messresultate konstant 100 MB beträgt. Dieses Gesamtdatenvolumen repräsentiert in seinem Umfang natürlich nicht die realen Mengenverhältnisse der zu migrierenden Datenbestände der DATEV eG. Da sich aber bei den Analysen zu [2] eine relativ gute Skalierbarkeit bezogen auf die ermittelten Performance-Ergebnisse zeigte, genügt bereits eine derartige Datenmenge zur Grundlage für die Exploration der real zu erwartenden Migrationskosten. Die angesprochene Regulierung der Gesamtdatenmenge erfolgt für die abgebildeten Messresultate ausgehend von der Merkmalsausprägung „1 KB x 100.000“ – also der Migration von 100.000 je 1 KB großen XML-Dokumenten – bis hin zur Ausprägung „10 MB x 10“

3.2 Analyse der Migrationskosten

Gemäß der eingangs angedeuteten Vorgehensweise erfolgt die Ermittlung der Performance der einzelnen Migrationsschritte durch die sukzessive Aufschlüsselung der Gesamtkosten der Migration in einzelne (Teil-)Prozessketten. Die anschließenden Ausführungen betrachten daher die folgenden Messreihen.

- CLOB-INSERT (Basiskosten)
- CLOB-INSERT (inklusive Unicode-Konvertierung)
- XML-INSERT (inklusive Unicode-Konvertierung und XML-Parsen)
- XML-INSERT (inklusive Konkatenation, Unicode-Konvertierung und XML-Parsen)

Die Performance des Vorverarbeitungsschritts wird aus den in Abschnitt 2.1 genannten Gründen nicht weiter untersucht. Die anschließend präsentierten Resultate zeigen die bei den Messreihen ermittelten CPU-Zeiten. Neben diesen sind bei der Performance-Analyse in DB2 auch diverse Elapsed-Zeiten (inklusive I/O-Zeiten, Lock/Latch-Waits, etc.) von Bedeutung, die aber aufgrund ihrer starken Schwankungen in Abhängigkeit zur aktuellen Last auf dem DB2-Entwicklungssystem der DATEV eG in diesem Beitrag vernachlässigt werden. Eine Gesamtaufgliederung der relevanten Zeiten findet sich in [2]. In den weiteren Ausführungen werden für die genannten Messreihen die Resultate aufgezeigt und teilweise andiskutiert. Umfangreichere Informationen dazu finden sich ebenfalls in [2].

Zuerst sei der Fokus auf die Zusatzkosten gerichtet, den die Schritte **Unicode-Konvertierung** und **XML-Parsen** verursachen. Hier zeigt sich in *Abbildung 6* eine mit zunehmender Größe der zu migrierenden XML-Dokumente generell fallende CPU-Zeit. Während die Zusatzkosten für die Unicode-Konvertierung verhältnismäßig gering ausfallen, resultiert für den zusätzlichen Anteil, den das XML-Parsen verursacht, ein wesentlich höherer Aufwand. Dieser ist jedoch der Komplexität des XML-Parse-Vorgangs geschuldet, bei dem neben der Überführung der Ausgangsdaten in die native Speicherform unter anderem auch deren Struktur geprüft wird.

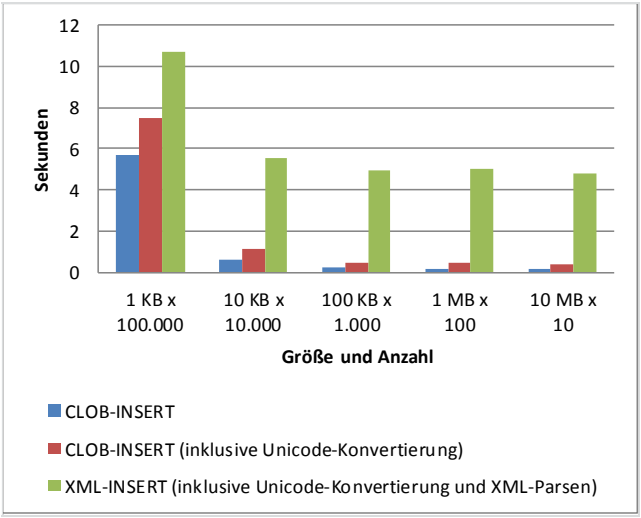

Abbildung 6: Performance der Migrationsschritte

Überraschenderweise sind die Grundkosten für einen einzelnen (komplexen) XML-Parse-Vorgang (unter anderem für das Laden und Initialisieren des XML-Parsers) jedoch kaum merklich, sodass sich die Messresultate ab der Merkmalsausprägung 10 KB x 10.000 trotz der kontinuierlich abnehmenden Anzahl von Dokumenten und somit auch XML-Parse-Vorgängen nur unwesentlich verändern. Mit anderen Worten formuliert bedeutet dies: Die relativ konstanten Gesamtkosten in *Abbildung 6* (jeweils circa 5 Sekunden CPU-Zeit für die "vier rechten hohen Balken") zeigen, dass der Kostenaufwand primär in den insgesamt zu parsenden Megabytes liegt, nicht aber in deren Aufteilung auf viele oder auf wenige XML-Dokumente.

Noch kostenintensiver als das XML-Parsen ist – eine gewisse Anzahl von Dokumentteilen vorausgesetzt – unerwarteterweise nur der Migrationsschritt der **Konkatenation**. Um dessen Einfluss geeignet zu visualisieren, sind die durch ihn bei der Datenerhebung zur Messreihe „XML-INSERT (inklusive Konkatenation, Unicode-Konvertierung und XML-Parsen)“ verursachten Kosten anteilig am Gesamtaufwand einer Migration in einer separaten *Abbildung 7* dargestellt.

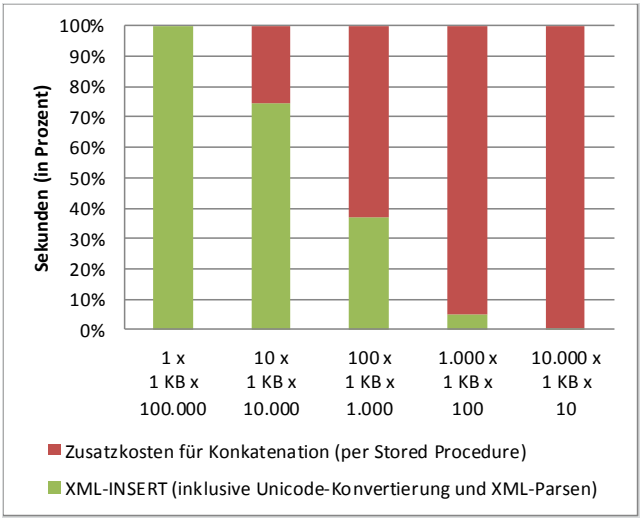

Abbildung 7: Kostenanteil der Konkatenation

Hier sei angenommen, dass stets 1 KB große Dokumentteile zu einem Gesamtdokument konkateniert und anschließend weitermigriert werden. Daher variiert bei den zu den abgebildeten Resultaten gehörenden Messreihen nicht mehr die Dokumentgröße in Abhängigkeit eines konstanten Gesamtdatenvolumens, sondern die Anzahl an, zu einem Dokument zu konkatenierenden Teile. Beispielweise werden bei der Merkmalsausprägung „10 x 1 KB x 10.000“ genau 10 je 1 KB große Dokumentteile zu einem 10 KB großen XML-Dokument zusammengefügt und anschließend wie ein „herkömmliches“ 10 KB großes nicht-natives XML-Dokument weitermigriert.

Auffällig ist hier, dass trotz der simplen Logik der Konkatenation mittels der CONCAT-Funktion dennoch ein erheblicher Aufwand entsteht, der ab einer Anzahl von 100 zu konkatenierenden Dokumentteilen den Hauptanteil der Gesamtkosten einer Migration einnimmt. An dieser Stelle lässt sich aber zumindest für die betrachteten Datenbestände der DATEV eG Entwarnung geben, da Konkatenationen mit derartig vielen Dokumentteilen im Rahmen der untersuchten Migration nicht vorzunehmen sind.

4. ZUSAMMENFASSUNG

Der vorliegende Beitrag hat einen Einblick in den Themenschwerpunkt der Migration von nicht-nativen XML-Dokumenten in den nativen XML-Datentyp in DB2 gegeben. Dabei erfolgte die Herleitung und Erarbeitung von zwingend erforderlichen und bedingt notwendigen Migrationsschritten sowie die Analyse deren Performance bezogen auf eine beispielhafte explorative Migration von CLOB-Daten mittels INSERT-Statements. Im Ergebnis dieser exemplarischen Auswertung zeigte sich, wie unterschiedlich die einzelnen Schritte die Gesamtkosten einer Migration beeinflussen und wo dabei die Hauptkostenfaktoren zu vermuten sind.

Neben dem fachlichen Aspekt soll dieser Beitrag aber ebenso dazu dienen, den wissenschaftlichen Charme eines praxisorientierten Anwendungsszenarios aufzuzeigen. Insbesondere wenn man wie in dem vorliegenden Beitrag angedeutet ins Detail einer konkreten Migrationslösung einsteigt, ergeben sich zahlreiche Parallelen zu anderen DBMS-Themengebieten wie Archivierung oder Aktive Datenbanken. Für den interessierten Leser seien daher zur vertieften Auseinandersetzung mit der hier diskutierten Thematik nochmals die Arbeiten [1] und [2] empfohlen. Insbesondere in [2] werden zudem auch weiterführende Untersuchungen zu den Themenschwerpunkten XML-Schema-Management (siehe auch [3]) und Query Performance angestellt.

5. LITERATUR

[1] C. Koch. *XML-Speicherstrukturen in DB2 for z/OS Version 9.1 und Überführungskonzepte für nicht-native XML-Speicherformen*, Studienarbeit, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena und DATEV eG, 08.2011.

[2] C. Koch. *Der Datentyp XML in DB2 for z/OS aus Performance-Perspektive*, Diplomarbeit, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena und DATEV eG, 01.2012.

[3] C. Koch. *Möglichkeiten und Konzepte zum XML-Schema-Management am Beispiel von DB2 for z/OS V9.1*, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena und DATEV eG, 06.2012.

[4] IBM Corporation. *DB2 pureXML – Intelligent XML database management*, Produkt-Homepage, 2012. http://www-01.ibm.com/software/data/db2/xml

[5] M. Päßler. *Datenbanken und XML – Passt das?*, 04.07.2007. http://inka.htw-berlin.de/datenverwaltung/docs/Paessler.pdf

[6] G. Lausen. *Datenbanken: Grundlagen und XML-Technologien*, Spektrum Akademischer Verlag, 1. Auflage, 2005.

[7] H. Schöning. *XML und Datenbanken: Konzepte und Systeme*, Carl Hanser Verlag GmbH & Co. KG, 1. Auflage, 2002.

[8] IBM Corporation. *DB2 Version 9.1 for z/OS – USER LOB VALUE STG field (LOBVALA subsystem parameter)*, 03.2011. http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2z9.doc.inst/src/tpc/db2z_ipf_lobvala.htm

[9] IBM Corporation. *DB2 Version 9.1 for z/OS – SYSTEM LOB VAL STG field (LOBVALS subsystem parameter)*, 03.2011. http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2z9.doc.inst/src/tpc/db2z_ipf_lobvals.htm

[10] IBM Corporation. *DB2 Version 9.1 for z/OS – XML Guide*, 12.2010. http://publibfp.dhe.ibm.com/epubs/pdf/dsnxgk16.pdf

[11] IBM Corporation. *DB2 Version 9.1 for z/OS – SQL Reference*, 03.2011. http://publib.boulder.ibm.com/epubs/pdf/dsnsqk1a.pdf

[12] IBM Corporation. z/OS – XML System Services User's Guide and Reference, 01.2012. http://www-03.ibm.com/systems/resources/gxlza151.pdf