erscheint in: Tagungsband der GI-Fachtagung 'Datenbanksysteme in Büro, Technik und Wissenschaft' (BTW'2001), A. Heuer (Hrsg.), Informatik aktuell, Oldenburg, März 2001, Springer-Verlag.

Ansätze der Nutzung von Erweiterbarkeitsmechanismen zur Anwendungsintegration in ORDBS - Eine qualitative und quantitative Evaluierung -

Marcus Flehmig, Henrik Loeser

Universität Kaiserslautern, AG DBIS,
Postfach 3049, D-67653 Kaiserslautern
e-mail: {flehmig | loeser}@informatik.uni-kl.de
http://wwwdbis.informatik.uni-kl.de/

Kurzfassung: Objekt-relationale Datenbanksysteme (ORDBS) bieten heute zahlreiche Erweiterungsmöglichkeiten. Über Erweiterungsschnittstellen ist es möglich, Teile der sonst in der separaten Anwendung vorhandenen Funktionalität in das ORDBS zu integrieren. Mittlerweile lassen sich sogar komplette Anwendungsdienste in Form eines Servers in das ORDBS integrieren, um so eine datennahe Verarbeitung zu erzielen. In diesem Beitrag wollen wir anhand einer an die TPC-W-Spezifikation angelehnten Anwendungsdomäne, einem elektronischen Geschäft (Web-Shop, e-shop), untersuchen, ob und welche Vorteile die Verlagerung von Anwendungsfunktionalität in das ORDBS bringt.

Schlüsselwörter: Web-Informationssystem, Erweiterbarkeit, ORDBS, TPC-W-Benchmark

1 Einleitung

Objekt-relationale Datenbanksysteme (ORDBS) bieten heute zahlreiche Erweiterungsmöglichkeiten, angefangen von benutzerdefinierten Typen ('*user-defined types*', UDTs) und Funktionen (UDFs) über abstrakte Tabellen bis hin zu benutzerdefinierten Indexstrukturen. Über Erweiterungsschnittstellen ist es möglich, Teile der sonst in der separaten Anwendung vorhandenen Funktionalität in das ORDBS zu integrieren und über die SQL-Schnittstelle anzusprechen. Mittlerweile lassen sich sogar komplette Anwendungsdienste in Form eines Servers in das ORDBS integrieren, um so eine datennahe Verarbeitung zu erzielen. Die Vorteile einer solchen Integration liegen in der Reduktion des Protokoll- und Kommunikationsaufwandes sowie in der leichteren Konsistenzsicherung zwischen Anwendungs- und Datenbankserver, z. B. im Hinblick auf Pufferungstechniken. Allerdings gibt es bisher wenig Erfahrungen und auch keine Entwurfsregeln für den Einsatz der objekt-relationalen Erweiterungsmöglichkeiten, insbesondere in Bezug auf die Integration von gesamten Anwendungsdiensten in ein ORDBS. Zudem weisen die verfügbaren ORDBS noch zahlreiche Einschränkungen auf.

Anhand einer konkreten Anwendungsdomäne wollen wir in diesem Beitrag untersuchen, ob die Dienste-Integration Vorteile bringt und worin diese möglicherweise liegen. Als Anwendungsgebiet betrachten wir dazu elektronische Geschäfte (Web-Shops, e-shops), speziell den im TPC-W [TPC00] spezifizierten Buchladen. Die vom TPC-W definierte Testumgebung für Web-basierte Verkaufssysteme wollen wir deshalb verwenden, weil hier ein realitätsnahes Szenario und geeignete Bewertungsmaßstäbe vorgegeben werden und zudem bereits erste Testprogramme (Lastgeneratoren) frei verfügbar sind.

Im nachfolgenden Kapitel stellen wir dazu die wichtigsten Merkmale des TPC-W-Benchmarks vor. In Kapitel 3 skizzieren wir die verwendeten Systemkomponenten und diskutieren mögliche Systemarchitekturen. Daran anschließend stellen wir die konkrete Messumgebung vor und präsentieren dann, in Kapitel 5, erste Messergebnisse, bewerten sie und erarbeiten auf der ORDBS-Integration beruhende Verbesserungen. Der Beitrag schließt mit einer Zusammenfassung sowie einem Ausblick auf die noch ausstehenden Arbeiten.

2 TPC-W-Benchmark-Spezifikation

Die jüngste Spezifikation des '*Transaction Processing Performance Council*', der TPC-Web, nimmt sich der Problematik der Leistungsbewertung von Web-basierten E-Commerce-Anwendungen an und berücksichtigt dabei auch transaktionale Gesichtspunkte [TPC00]. Die Spezifikation beschreibt ein typisches Web-basiertes Anwendungsszenario auf der Basis eines elektronischen Buchhandels ('*book stores*'). Drei Aspekte des TPC-W erscheinen für sein Verständnis besonders wichtig: die Software-Komponenten (siehe Abb. 1), die '*Book Store*'-Applikation mit den ihr zugehörigen Web-Interaktionen (Stöbern, Suchen, Bestellen usw.) und die zugrunde liegende Datenbasis.

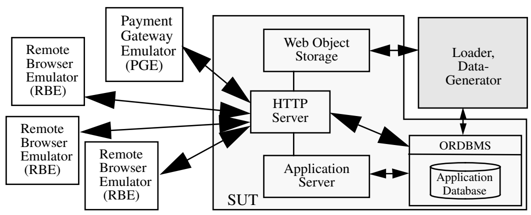

Abb. 1: SW-Komponenten des TPC-W-Szenarios

2.1 Software-Komponenten: RBE, SUT

Der sog. '*Remote Browser Emulator*' (RBE) emuliert das typische Verhalten eines Benutzers. Basierend auf vorgegebenen Wahrscheinlichkeiten werden HTTP-Anfragen für die jeweiligen Web-Interaktionen generiert, die Antworten analysiert, daraufhin neue Anfragen generiert und die Antwortzeiten gemessen. Das zu testende System ('*system under test*', SUT) besteht aus den in Abb. 1 gezeigten Komponenten. Eingehende Anfragen werden vom HTTP-Server verarbeitet und, wenn nötig, an den Anwendungsserver weitergeleitet. Dabei können Strategien zur Lastbalancierung eingesetzt werden. Die Anwendungslogik kann sich sowohl im DBS als auch im Anwendungsserver befinden.

2.2 TPC-W-Beispielanwendung

Die 'Book Store'-Applikation unterteilt sich in vierzehn der sog. Web-Interaktionen. Drei unterschiedliche Web-Interaktionsmixe für die Simulation eines hauptsächlich durch Stöbern ('*browsing*'), gelegentliches Bestellen ('*shopping*') oder häufiges Bestellen ('*ordering*') geprägten Benutzerverhaltens werden definiert. Sie unterscheiden sich in der Gewichtung des nur lesenden und bestellbezogenen Anteils ('*browsing*' und '*ordering*') und resultieren in drei Metriken zur Beschreibung des Leistungsverhaltens. Die Web-Interaktionen pro Sekunde (WIPS) beschreiben die Leistung innerhalb eines '*Shopping Interaction Mixes*', WIPSb und WIPSo das Leistungsverhalten der anderen Mixe.

2.3 TPC-W-Datenbasis: DB-Skalierung und -Konfiguration

Das der TPC-W-Spezifikation zugrunde liegende DB-Schema ist ein einfaches relationales Schema und besteht aus acht Tabellen. In diesen werden die Daten zu Kunden, Adressen, Länderinformationen, Bestellungen sowie Bestellpositionen, Produkte ('*item*'), Autoren und Kreditkarten-Transaktionen ('*cc_xacts*') verwaltet. Die Skalierung der Datenbank wird durch zwei Parameter beeinflusst. Die Anzahl der emulierten Browser (EBs) und die Kardinalität der Produktdatenrelation ('*item*') bestimmen die initiale DB-Konfiguration. Diese beiden Parameter bilden die zwei Startwerte zur sukzessiven Berechnung weiterer Kardinalitäten (siehe Tabelle 1). Die Anzahl der zu betrachtenden Länder ist fest vorgegeben. Die übrigen Relationen müssen gemäß der Vorgaben vor der ersten Messung entsprechend gefüllt sein. Bei weiteren Messungen kann die Datenbank jeweils in den Initialzustand zurückgesetzt werden.

Table Name	**Cardinality (in rows)**
CUSTOMER	2880 * (number of EB)
COUNTRY	92 (fix)
ADDRESS	2 * CUSTOMER
ORDERS	0.9 * CUSTOMER
ORDER_LINE	3 * ORDERS
AUTHOR	0.25 * ITEM
CC_XACTS	1 * ORDERS
ITEM	1K, 10K, 100K, 1M, 10M

Tabelle 1: DB-Konfiguration

3 Komponenten und Architekturen

Nach der Übersicht über den TPC-W-Benchmark stellen wir im Folgenden die in der Regel zur Realisierung eines Web-Shops eingesetzten Komponenten genauer vor und diskutieren mögliche Systemarchitekturen, d. h. die Gruppierung von Komponenten zu Programmen und deren Verteilung auf Rechner.

3.1 Komponenten

Zum Erzeugen und zur Verarbeitung von Ressourcenanforderungen, d. h. das Anfordern, Erstellen und Ausliefern von Web-Seiten, kommen bei unseren Tests analog zum oben beschriebenen TPC-W-Benchmark vier große Komponenten zum Einsatz: RBE, HTTP-Server, Anwendungsserver, DBS. Der HTTP-Server greift typischerweise auf in einem Dateisystem gespeicherte Dateien oder auf andere, z. T. weit entfernte Dienste zu. Letztere können in Form von Erweiterungsmodulen des HTTP-Servers, separaten

Programmen oder (entfernten) eigenständigen Servern realisiert sein. Zusätzlich zu den beschriebenen vier werden noch zahlreiche kleinere Module benötigt. Sie handhaben u. a. die Kommunikation zwischen den Komponenten und ermöglichen den DB-Zugriff. In der Regel sind sie in Form von Bibliotheken in die größeren Module integriert.

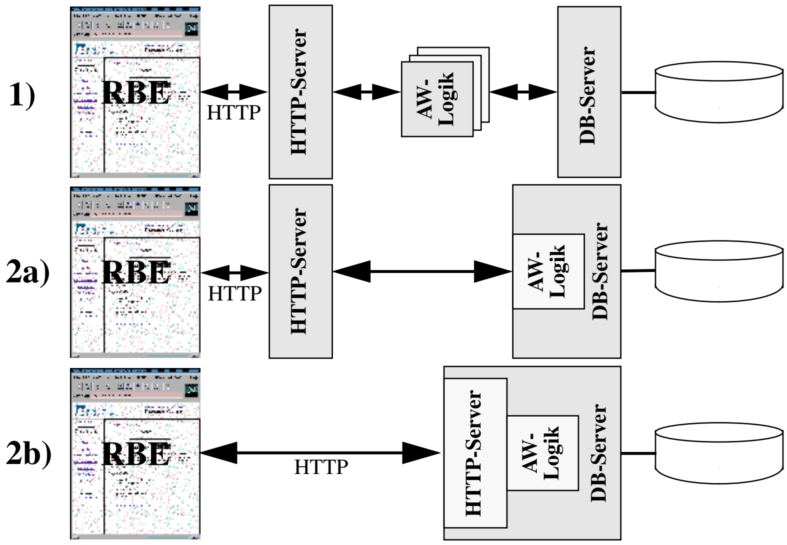

Abb. 2: Systemarchitekturen

3.2 Systemarchitekturen

Die oben vorgestellten Komponenten lassen sich nun aus Leistungssgründen auf mehrere Rechner verteilen. Dabei kann die Anwendungslogik sowohl unterschiedlichen Komponenten zugeordnet als auch diese mit Hilfe verschiedener Techniken miteinander verknüpft, an den HTTP-Server angebunden und, je nach Systemarchitektur, in das ORDBS integriert werden. Im Folgenden gehen wir kurz auf die von uns betrachteten Formen der Anbindung ein (siehe Abb. 2).

1. *Anwendungsserver*: Die entsprechende Funktionalität wird in einem eigenständigen Dienst realisiert, der auf eine separate Maschine verlagert werden kann. Der Anwendungsserver erzeugt ein entsprechendes Antwortdokument. Hierfür greift er auf das ORDBS zu.
2. *Volle Integration*: Die gesamte Anwendungsfunktionalität wird in Form eines Anwendungsservers (2a) oder eines HTTP-Servers mit Erweiterungsmodulen (2b) in das ORDBS integriert. Im Fall 2a) leitet der HTTP-Server, anstatt ein CGI-Programm oder ein Erweiterungsmodul aufzurufen, eingehende Ressourcenanforderungen direkt an den eingebetteten Server weiter.

Andere Systemarchitekturen wurden aus Gründen der Vergleichbarkeit von uns nicht berücksichtigt, da sie eine Re-Implementierung der TPC-W-Beispielanwendung erfordern würde. Die Beschränkung auf die oben genannten Architekturen hat den Vorteil, dass die TPC-W-Anwendung ohne nennenswerte Anpassungen in den verschiedenen

Architekturen eingesetzt werden kann. Es kann infolgedessen direkt auf den quantitativen Nutzen einer Anwendungsintegration geschlossen werden.

4 Messumgebung

Um das Leistungsverhalten beurteilen zu können, werden auf der Basis der TPC-W-Benchmark-Spezifikation Messungen durchgeführt, die einen Vergleich zwischen externer Ausführung (Variante 1) und interner bzw. server-integrierter Ausführung (Variante 2) erlauben. In den folgenden Abschnitten wird zunächst die eingesetzte Infrastruktur beschrieben und auf TPC-W-relevante Aspekte eingegangen. Für allgemeine Kriterien der Leistungsbewertung sei an dieser Stelle auf [Sch99] oder [LR00] verwiesen. Hinweise zu TPC-W-bezogenen Aspekten hingegen finden sich unter [TPC00] oder [PF00]. Konzepte und Ergebnisse zur Abbildung und Realisierung der TPC-W-Beispielanwendung auf objekt-orientierten DBS, sowie Hinweise zur Pufferung von Objekten sind in [Gup00] beschrieben.

4.1 Infrastruktur und Rechensysteme

Um ein möglichst reales Anwendungsszenario nachzubilden, werden Client (RBE), Web-Frontend (HTTP-Server) und Backend (Anwendungsserver und DBS) auf jeweils separaten Maschine betrieben. Der Client befindet sich auf einer SUN Ultra 2 mit 2 Prozessoren (300 MHz / 1 GByte RAM), das Web-Frontend auf einer SparcStation 20 mit einem Prozessor (60 MHz / 132 MByte RAM) und das Backend-System auf einer SUN Enterprise 450 mit 4 Prozessoren (300 MHz / 1 GByte RAM). Es handelt sich ausschließlich um Solaris V5.7 Systeme mit 10 MBit-Netzanbindung.

Das Web-Frontend dient neben dem Bereitstellen eines HTTP-Dienstes auch der Lastbalancierung. Die eher moderate Rechenleistung stellt hier kein einschränkendes Kriterium für den maximalen Leistungsdurchsatz dar, denn alle Anfragen werden an das Backend-System propagiert. Aufgrund der Vergleichbarkeit der Messungen verschiedener Architekturvarianten sind Anwendungsserver (Servlet-Engine) und das ORDBS zu einem System (Backend) zusammengefasst.

4.2 TPC-W-Aspekte

Die Trennung des HTTP-Servers von der Servlet-Engine hat den Vorteil, leistungsstarke HTTP-Server verwenden zu können, die auch in vielen anderen kommerziellen Systemen eingesetzt werden. Der Einsatz eines HTTP-Servers mit integrierter Servlet-Engine innerhalb eines DBS unter Benutzung von Java würde aber einen HTTP-Server erfordern, der wiederum auch in Java implementiert ist. Auch wenn Java-HTTP-Server bereits sehr leistungsstark sind, so sind sie Realisierungen in anderen Sprachen (C++) noch immer unterlegen [W3C98, W3C99]. Die entsprechende Architekturvariante wurde daher nicht weiter betrachtet. Einsatz finden vielmehr der sehr weit verbreitete Apache HTTP-Server (V1.3.12, [Apa00a]) sowie die zugehörige Servlet-Engine (Tomcat 3.2, [Apa00b]). Zusammen mit einem kommerziellen ORDBS[1], welches die Möglichkeit bietet, ganze Java-Anwendungen DBS-intern zu verarbeiten, bilden sie das SUT.

Der RBE und auch die '*Book Store*'-Applikation sind komplett in Java geschrieben und entstammen der frei verfügbaren TPC-W-Suite der PHARM-Gruppe der University of Wisconsin-Madison [PHA99], die nahezu unverändert übernommen wurde und alle spezifizierten Web-Interaktionen durch je ein eigenes Servlet realisiert. Das DB-Schema wurde durch einen selbst realisierten Generator in zwei unterschiedlichen Konfigurationen mit Daten gefüllt. Die Anzahl der emulierten Browser für die initiale DB-Konfiguration variierte zwischen den beiden Werten 10 und 30, die der Produktdaten betrug 10.000. Für die häufigsten Zugriffspfade wurden Indexstrukturen definiert. Fragmentierungsaspekte wurden zwar zunächst berücksichtigt, erwiesen sich aber bei den oben genannten Konfigurationen als nicht notwendig. Es wurde allein der '*Browsing Web Interaction Mix*' (WIPSb) betrachtet. Dessen Verwendung erlaubt die Variation der Anzahl der emulierten Browser bei unterschiedlichen Messungen unabhägig von der initialen DB-Konfiguration. Die Dauer eines Messintervalls betrug jeweils 30 Minuten.

Zwecks besserer Ausnutzung des Mehrprozessorbetriebs wurden drei Servlet-Engine-Instanzen bei den Messungen verwendet. Die drei Servlet-Engines konnten unverändert auch DBS-intern eingesetzt werden, wohingegen die DB-Komponente der Web-Anwendung in nur einem Aspekt eine Anpassung erfahren musste, um eine DBS-interne Kommunikation zu ermöglichen. DBS-intern liefen die Server-Prozesse innerhalb einer eigenen '*Java Virtual Machine'* (JVM) und konnten durch UDFs [SQLJ99] gestartet, administriert und beendet werden.

5 Bewertung und Verbesserungen

In der vorgestellten Umgebung wurden sowohl unter Verwendung eines externen als auch mit einem in das ORDBS integrierten Anwendungsserver Messungen durchgeführt. Im Folgenden stellen wir erst die Ergebnisse vor, anschließend bewerten und diskutieren wir sie. Danach wollen wir, basierend auf den Ergebnissen, durch die Integration des Anwendungs- in den DB-Server entstehende Verbesserungsmöglichkeiten erörtern.

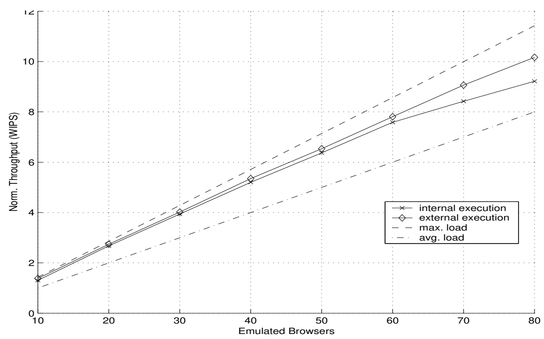

Abb. 3: Messung mit großer DB-Konfiguration (30 EBs)

5.1 Messergebnisse

Es wurden Messungen unter Berücksichtigung unterschiedlicher Skalierungen durchgeführt, wovon wir zwei exemplarisch aufgreifen. Abb. 3 zeigt den Vergleich zwischen interner und externer Ausführung in der Konfiguration für 30 EBs, also für 86.400 Kunden und 10.000 Produkte. Der Datendurchsatz ist nahezu gleich hoch. Allerdings

1. Eine Nennung des ORDBS-Herstellers ist aus lizenzrechtlichen Gründen leider nicht möglich.

erreicht die interne Verarbeitung im Bereich von 70 bzw. 80 EBs nicht mehr die Durchsatzrate der externen Verarbeitung. Der theoretische maximale Durchsatz, den der RBE zu erzeugen in der Lage ist, wird von beiden Lösungen nicht erreicht.

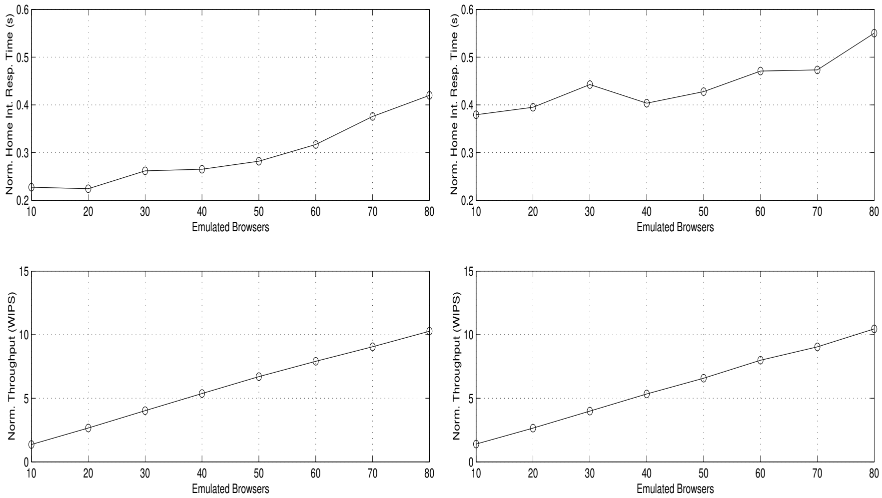

Abb. 4: Messung mit kleiner DB-Konfiguration (10 EBs)

Wie man den in Abb. 4 für eine kleine Skalierung (28.000 Kunden / 10.000 Produkte) gezeigten Messergebnissen entnehmen kann, ist das Leistungsverhalten unter bestimmten Bedingungen beim Einsatz der internen wie auch der externen Verarbeitungsvariante nahezu gleich. Dabei ist jedoch zu beobachten, dass die Antwortzeiten bei der internen Verarbeitung ein wenig geringer sind (siehe obere Grafiken). Die linke Grafik zeigt die Ergebnisse für die interne, die rechte für die externe Verarbeitung.

5.2 Bewertung

Die Messergebnisse zeigen für die beiden betrachteten Systemarchitekturen annähernd gleiche Leistungen beim Durchsatz, d. h. der Zahl der WIPS. Als einziger Unterschied sind beim integrierten Anwendungsserver kürzere Antwortzeiten festzustellen, was auf einen schnelleren Datenaustausch zwischen ORDBS und Anwendungsserver schließen lässt. Unklar ist, warum die kürzeren Antwortzeiten insgesamt nicht in einem höheren Durchsatz resultieren. Als Gründe kommen die (programmierten) Wartezeiten seitens des RBE und möglicherweise die damit verbundene Nichtauslastung des Anwendungsservers in Frage. Diesbezüglich bedarf es also noch der Klärung.

Der Leistungseinbruch bei der internen Verarbeitung in der größeren Konfiguration lässt sich mit der herstellerspezifischen Realisierung der Integrationsschnittstelle erklären. Sowohl die Anfrageverarbeitung als auch das Ausführen von Java-Routinen, d. h. der Anwendungslogik, finden in einem gemeinsamen Betriebssystem-Prozess statt, um die Möglichkeiten der DBS-internen Kommunikation voll auszuschöpfen, ohne hohen Realisierungsaufwand betreiben zu müssen. Parallelität ist damit aber nur über den Einsatz mehrerer paralleler Anwendungsserver-Instanzen (DBS-intern) möglich. Bei externer Verarbeitung können aber schon mit einer Anwendungsserver-Instanz Anfragen auf der DB-Ebene parallel ausgeführt werden. Mehrere Anwendungsserver-Instanzen verstärken den Grad der Parallelität weiter. Ferner lassen sich in einer Mehrprozessorumgebung bei einer externen Verarbeitung die verschiedenen Prozesse besser auf die physischen Prozessoren verteilen. Der Vorteil der DBS-internen Kommunikation allein kann diesen Nachteil nicht aufheben.

Berücksichtigt man, dass die Möglichkeiten zur Integration eines Anwendungsservers noch „jung“ sind und die entsprechende Realisierung daher noch verbesserungsfähig ist, so ist es schon als positiv zu vermerken, dass durch die Integration in das ORDBS keine (deutlichen) Verschlechterungen auftreten. Durch die Wahl der Sprache Java erfolgte die Integration im Wesentlichen durch den Austausch des JDBC-Treibers und den initialen Aufruf des Anwendungsservers als UDF, am HTTP-Server mussten keine Änderungen vorgenommen werden. Insgesamt ist also der Umstellungsaufwand von einer externen zu einer integrierten Lösung gering, so dass ein Wechsel schnell vonstatten geht.

5.3 Mögliche Verbesserungen

Da sich die Leistungen der beiden unterschiedlichen Lösungen nur in der Antwortzeit unterscheiden, stellt sich die generelle Frage, wie durch die ORDBS-Integration für die untersuchte Anwendungsdomäne Leistungssteigerungen erzielt werden können. Erste Ansätze können sich z. B. durch die Nutzung der engen Integration für eine kohärente Pufferung von Anfrageergebnissen bzw. Fragmenten von Antwortseiten ergeben. So fallen bei genauer Betrachtung der im Rahmen des TPC-W spezifizierten DB-Anweisungen die für die Verkaufsförderung (*promotional*) verwendeten Anfragen auf. Sie erzeugen für einzelne Kategorien und Bücher oder auf globaler Ebene sortierte Listen mit Buchvorschlägen. Da diese Listen in jede generierte Seite eingebunden werden müssen, ist ihre Zwischenspeicherung als entsprechend vorgenerierte Dokumentenfragmente sinnvoll. Dabei muss sichergestellt werden, dass trotz einer Pufferung noch die jeweils aktuellen Daten in die generierten Dokumente eingebunden werden. Hierzu lassen sich die von ORDBS angebotenen Trigger nutzen, die nach einer Änderung relevanter Daten über einen UDF-Aufruf als Aktion die gepufferten Inhalte invalidieren.

Caching von Fragmenten und Invalidierung

Über die Möglichkeiten der Dienste-Integration in das ORDBS lässt sich ein entsprechender Cache für die verwandten Artikel, Bestseller und Neuerscheinungen integrieren (siehe Abb. 5). Mit Hilfe von DB-Triggern kann nach entsprechenden Änderungsoperationen eine Methode des Cache aufgerufen werden, um Inhalte, d. h. die relevanten Dokumentenfragmente zu invalidieren.

Auf diese Weise lassen sich ohne Probleme die sonst bei jeder Web-Interaktion zu erzeugenden Fragmente im Puffer vorhalten, ohne dass es zu Konsistenzproblemen kommt, die geforderten Transaktionseigenschaften werden erfüllt. Über die Pufferung und die hierdurch vermiedene Anfrageverarbeitung ist es möglich, die Antwortzeiten zu verkürzen und vermutlich einen höheren Durchsatz zu erzielen. Der vorgestellte Ansatz ist ähnlich der in [CID99] beschriebenen Systemarchitektur. Hier werden jedoch die verwendeten Komponenten (sog. ‘*Trigger Table’*, ‘*Cache Manager’* etc.) als separate externe Anwendungsdienste und nicht als in das ORDBS integrierte Module realisiert.

```
public class RelatedItemCache {
// a simple cache
        private static Hashtable cachedData;
                public static void invalidateCache() {
                // invalidate the Cache
        }
        public static Vector getRelatedItems(long item_id) {
// get the five related items together with their image data
                Long key = new Long(item_id);
                if (cachedData.contains(key)
                        Vector cachedData = (Vector) cachedData.get(key);
                        return cachedData;
                } else {
// Open direct DB connection and get Vector with related items
// put it into the cache and return new vector
                        return newData;
                }
        }
}
create procedure inv_cache()
external name 'RelatedItemCache.invalidateCache()'
language java;
create trigger t01 update on items
after (execute procedure inv_cache());
```

Abb. 5: Integrierter Cache mit Trigger-basierter Invalidierung

DB-gesteuerte Cache-Aktualisierung

Weitere Verbesserungen lassen sich über die automatische Aktualisierung der im Cache verwalteten Fragmente erzielen. So ist es z. B. möglich, die Listen mit Kategorie-abhängigen Neuerscheinungen nach dem Einfügen neuer Artikel erneut zu generieren und dann im Cache abzulegen. Hierzu kommt das in [Loe00] vorgestellte Verfahren der ORDBS-gesteuerten Dokumentengenerierung zum Einsatz, nur mit dem Unterschied, dass die erzeugten Dokumente bzw. Fragmente nicht im Dateisystem oder im DBS, sondern im Cache abgelegt werden.

6 Zusammenfassung und Ausblick

In diesem Beitrag haben wir für eine konkrete Anwendungsdomäne, ein elektronisches Verkaufssystem, untersucht, ob die Integration eines Anwendungsdienstes in ein objekt-relationales DBS im Vergleich zu einem externen Dienst Vorteile bringt. Um hierfür schnell eine realitätsnahe Messumgebung einsetzen zu können, wurde der TPC-W-Benchmark verwendet. Die Integration des Anwendungsservers erfolgte durch den einfachen Austausch des JDBC-Treibers sowie durch das initiale UDF-basierte Starten des Dienstes. Ansonsten wurden keine Änderungen vorgenommen, so dass der Integrationsaufwand als sehr gering bezeichnet werden kann. Bei unseren ersten Tests zeigte sich, dass trotz des z. T. doch erst prototypischen Zustands der Erweiterungsschnittstellen des verwendeten ORDBS bei einer Integration des Anwendungsservers im Vergleich zu einem externen Anwendungsserver keine Leistungseinbußen auftreten. Nur bei sehr großer Last waren die derzeitigen Einschränkungen deutlich erkennbar.

Im Anschluss an die Diskussion der Messergebnisse haben wir mögliche Verbesserungen der integrierten Lösung diskutiert. Diesbezüglich erscheint besonders die Pufferung von Zwischenergebnissen bzw. Dokumentfragmenten und die Nutzung von Triggern für ihre Invalidierung bzw. Aktualisierung sehr interessant. Die diskutierten Verbesserungsmöglichkeiten bedürfen allerdings noch einer genaueren Evaluation in der Praxis. Neben weiteren Tests in der in diesem Beitrag vorgestellten Umgebung müssen aber auch noch andere verfügbare ORDBS bezüglich der Integration von Anwendungsdiensten untersucht werden. Aufgrund der stetig zunehmenden Bedeutung der objekt-relationalen DB-Technologie und der immer noch fehlenden Erfahrungen mit dem Einsatz der ORDBS-Erweiterungsschnittstellen halten wir dies auch für die Zukunft für ein interessantes Forschungsthema, das es zu behandeln gilt.

Literatur

[Apa00a] Apache Software Foundation: *Apache Web Server*, http://www.apache.org, 2000.

[Apa00b] Apache Software Foundation: *Jakarta/Tomcat*, http://jakarta.apache.org, 2000.

[CID99] J. Challenger, A. Iyengar, P. Dantzig: *A Scalable System for Consistently Caching Dynamic Web Data*, Proceedings of IEEE INFOCOM'99, New York, USA, März 1999.

[Gup00] M. Gupta: *TPC-W E-Commerce Benchmark Using Javlin/ObjectStore*, IEEE Bulletin of the Technical Committee on Data Engineering Volume 23(1): 43-48, März, 2000

[Loe00] H. Loeser: *Keeping Web Pages Up-To-Date With SQL:1999*, Int. Database Engineering and Applications Symposium (IDEAS 2000), pp. 219-223, Yokohama, Japan, September 2000.

[LR00] A. Labrinidis, N. Roussopoulos: *Generating Dynamic Content at Database-backed Web Servers: cgi_bin vs mod_perl*, ACM SIGMOD Record Volume 29(1): 26-31, 2000

[Sch99] A. Schmietendorf: *Leistungsbewertung von Web-Applikationen*,
Vortrag im Rahmen der 12. CECMG Jahrestagung, Bremen, Deutschland,
http://www.uni-magdeburg.de/schmiete/diplom/jtcecmg99.htm, 1999

[SQLJ99] *SQLJ: SQL Routines using the Java Programming Language,*
http://www.sqlj.org, June 18, 1999.

[TPC00] Transaction Processing Performance Council: *TPC-W Benchmark Specification*,
http://www.tpc.org, 2000.

[PF00] M. Poess, C. Floyd: *New TPC Benchmarks for Decision Support and Web Commerce*,
ACM SIGMOD Record Web Edition, 2000

[PHA99] University of Wisconsin-Madison, Electrical & Computer Engineering
(resp. Prof. Lipasti's Fall 1999 ECE 902 course),
http://www.ece.wisc.edu/~pharm/tpcw.shtml, 1999.

[W3C98] World Wide Web Consortium: *Jigsaw Performance Evaluation*,
http://www.w3.org/Jigsaw/User/Introduction/performance.html, May 27, 1998.

[W3C99] World Wide Web Consortium: *HTTP Performance Overview*,
http://www.w3.org/Protocols/HTTP/Performance/, October 26, 1999.