HADES – Ein hochverfügbares verteiltes Main-Memory DBMS für eventbasierte Systeme

Matthias Meixner, Alejandro Buchmann

meixner@informatik.tu-darmstadt.de, buchmann@informatik.tu-darmstadt.de

Technische Universität Darmstadt

Abstract: Dieser Beitrag beschreibt, wie durch den Einsatz von Fehlertoleranz Festplatten durch eine schnellere aber fehleranfälligere Technologie ersetzt werden können, um die Geschwindigkeit von Datenbanken in eventbasierten Systemen zu steigern.

1 Einführung und Grundlagen

Während sich in den letzten Jahren die Speicherkapazität dramatisch erhöht hat, konnte die Zugriffszeit von Festplatten kaum verbessert werden. Neue Anwendungen erfordern aber niedrige Zugriffszeiten, die mit dieser Technologie nicht mehr erreicht werden können.

Zu diesen Anwendungen gehören eventbasierte Systeme [CRW01] [FMG03]. In eventbasierten Systemen kommunizieren verschiedene Komponenten über *Notifications (Benachrichtigungen)*, die das Auftreten eines bestimmten Ereignisses signalisieren. Diese werden im *Event Notification System* von *Event-Brokern* von den Produzenten zu den Konsumenten weitergeleitet und dabei bereits so gefiltert und transformiert, daß Informationen nur an die Stellen weitergeleitet werden, an denen es auch Konsumenten dafür gibt. Zur Vereinfachung sollen im folgenden beliebige Transformationen als Filter bezeichnet werden.

Da in diesem Modell weder die Identität noch die Anzahl der Konsumenten bekannt ist, kann keine Ende-zu-Ende Fehlerkorrektur eingesetzt werden und daher kann nicht garantiert werden, daß bei einem Ausfall einer Komponente keine Nachricht verloren geht.

Dieses Verhalten ist für Anwendungen nicht tolerabel, die erfordern, daß Benachrichtigungen exakt einmal übertragen werden. Dieses Problem läßt sich durch den Einsatz von Transaktionen und persistenter Speicherung also durch den Einsatz von Datenbankmechanismen lösen: Benachrichtigungen werden weitergegeben, indem sie innerhalb einer (verteilten) Transaktion von einer Datenbank in die nächste übertragen werden. Die persistente Speicherung schützt dabei vor einem Datenverlust, wenn ein Rechner ausfällt.

Durch die persistente Speicherung wird es erstmals auch möglich, zuverlässig den Inhalt mehrere Benachrichtigungen (Events) zu aggregieren und zu einer neuen Benachrichtigung zusammenzufassen und somit Filter über die „Zeit“ zu unterstützen.

Durch dieses Vorgehen ergeben sich jedoch auch Probleme: Die Zeit, die für ein Commit benötigt wird, limitiert die Ende-zu-Ende Laufzeit, da der nächste Filter Daten erst dann lesen kann, wenn das Commit des vorangegangenen Filters abgeschlossen ist. Fällt eine Datenbank aus, so gehen zwar keine Benachrichtigungen verloren, aber der Datenfluß ist unterbrochen. Deshalb wird eine Datenbank benötigt, die sehr schnelle Transaktionen bietet und eine hohe Verfügbarkeit gewährleistet.

Flaschenhals beim Commit ist der zum Logging benötigte Zugriff [GR93] auf die Festplatte, der die insgesamt benötigte Zeit dominiert: Selbst bei den schnellsten Platten (IBM Ultrastar, 15.000 U/min) fallen durchschnittlich $3,4ms$ Positionierzeit und $2ms$ Latenz an. Auch Main-Memory Datenbanken wie z.B. Prisma/DB [AvdBF+92] verwenden Festplatten für das Logging und können das Problem daher nicht lösen.

RAID-Systeme können zwar den Durchsatz, nicht aber die hier entscheidende Zugriffszeit verbessern. Solid-State-Disks (SSD) bieten zwar niedrige Zugriffszeiten, sind aber entweder nicht für schreibintensive Anwendungen geeignet (Flash-ROM basiert) oder für diesen Einsatz zu teuer (DRAM basiert, ca. 8000,- EUR für 800MBytes).

2 HADES Architektur

HADES (**H**ighly **a**vailable distributed main-memory **d**atabase for **e**vent based **s**ystems) verfolgt daher die Idee, Persistenz nicht durch den Einsatz von externen Speichermedien, sondern durch den Einsatz von Redundanz zu garantieren. Daten werden im Hauptspeicher von mehreren Rechnern eines Clusters gespeichert. Bei einem Rechnerausfall kann so noch weiterhin ohne Unterbrechung auf eine Kopie zugegriffen werden. Im Vergleich zu einer Speicherung auf Festplatte verspricht dies insgesamt eine höhere Verfügbarkeit und geringere Zugriffszeiten, die nur durch die niedrigere Latenz der Netzwerktechnologie begrenzt wird. Darüberhinaus bieten sich weitere Vorteile: Der Zugriff auf Daten ist nicht an eine feste Blockgröße gebunden, sondern die Größe der über eine Seitennummer adressierten Speicherbereiche kann gleich der Datensatzgröße gewählt werden.

Das Fehlermodell, das hier zugrunde gelegt werden soll ist *Crash* [Gär01], d.h. entweder ein Rechner arbeitet fehlerfrei oder überhaupt nicht. Während der Wiederherstellung der Redundanz nach einem Ausfall, darf kein weiterer Rechner ausfallen. Eine unterbrechungsfreie Stromversorgung verhindert, daß bei einem Stromausfall gleichzeitig alle Rechner ausfallen, das Netzwerk ist redundant ausgelegt, um eine Teilung zu verhindern.

Die redundante Speicherung wird in HADES dadurch realisiert, daß Daten in einer zweistufigen Abbildung im Cluster verteilt werden: Im ersten Schritt wird die Seitennummer auf die *Slice-Adresse* abgebildet: $Slice = Seitennummer \bmod SliceAnzahl$
Die Slice-Anzahl ist eine konstante Größe, die sich zur Laufzeit nicht ändert. Dies entspricht einer horizontalen Fragmentierung der Daten [SK98]

Im zweiten Schritt werden jedem Slice zwei Rechner zugeordnet (Primär- und Sekundärserver). Der Primärserver koordiniert die Zugriffe auf die Daten und den Abgleich mit der Kopie auf dem Sekundärserver, um das Auftreten von Inkonsistenzen zu verhindern. Der Sekundärserver kann als exakte Kopie des Primärservers bei einem Ausfall dessen

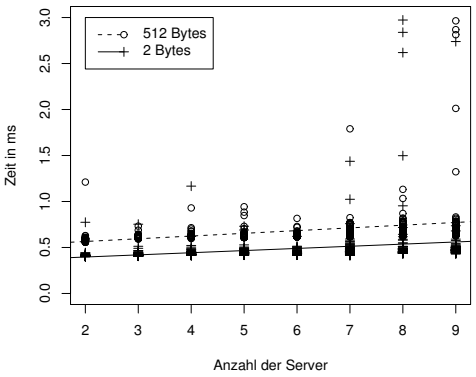

Abbildung 1: Schreiben von Daten

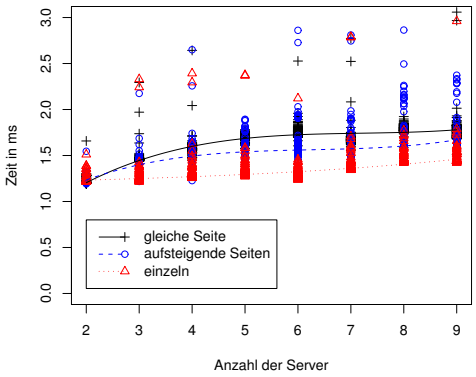

Abbildung 2: Lesen - modifizieren - schreiben

Funktion nahtlos übernehmen.

Fällt ein Server aus oder wird ein Server hinzugefügt, wird diese Zuordnung von einem ausgewählten Koordinator neu erstellt, um die Last wieder gleichmäßig zu verteilen und ausgefallene Server zu ersetzen. Dabei wird die vorangegangene Verteilung berücksichtigt, um die Menge der Daten, die umkopiert werden muß, zu reduzieren.

HADES stellt zur Koordination mehrerer Clients Transaktionen mit den klassischen *ACID-Eigenschaften* (Atomicity, Consistency, Isolation und Durability [WV02]) zur Verfügung. Diese werden von einem Koordinator verwaltet. Ein Backup-Koordinator übernimmt bei einem Ausfall dessen Aufgaben. Anstelle von Logging verwendet HADES Shadow-Paging [GR93]. Durch die Wahl der Datensatzgröße als Seitengröße vermeidet HADES die Probleme von festplattenbasierten Systemen bei einem gleichzeitigen Zugriff unterschiedlicher Transaktionen auf Datensätze, die zufällig im gleichen Block gespeichert sind [SK98]. Neben einfachen Transaktionen werden auch verteilte Transaktionen über mehrere Datenbanken hinweg unterstützt.

3 Performance-Messungen

Im folgenden werden Messungen für typische Operationen auf einem Linux-Cluster aus 10 identischen mit Fast-Ethernet gekoppelten AMD Athlon XP2000+ Rechnern vorgestellt.

Schreiben von Daten
Diese Operation wurde mit Seitengrößen von 2 und 512 Bytes und mit einer unterschiedlichen Anzahl von Servern durchgeführt. Die Messung mit 512 Bytes kann direkt mit Festplattenzugriffen verglichen werden. HADES bleibt mit Zugriffszeiten um $0,5ms$ etwa eine Größenordnung unter der Zugriffszeit von schnellen Festplatten (Abbildung 1).

Transaktionen
Hierfür wurde eine Transaktion bestehend aus Lesen, Verändern und Zurückschreiben gewählt, die dem in der Einführung verwendeten Szenario zur Weitergabe von Benach-

richtigungen entspricht. Es wurden drei Varianten verwendet, die sich darin unterscheiden, ob auf noch nicht freigegebene Locks gewartet werden muß oder ob noch Rechenzeit zum Abschluß vorangegangener Operationen benötigt wird: Es wird immer der gleiche Datensatz gelesen, immer ein anderer oder eine Pause zwischen den Zugriffen gelassen. Mit Zeiten um $1,5ms$ für die komplette Transaktion bleibt HADES deutlich unter der Zugriffszeit von Festplatten (Abbildung 2).

4 Fazit

HADES unterschreitet deutlich die Zugriffszeit von Festplatten und kann somit die Zugriffszeiten von etablierten Datenbanken und insbesondere auch Main-Memory Datenbanken wie z.B. PRISMA/DB [AvdBF+92] in der Zugriffszeit unterbieten, die beide aufgrund des Loggings von der Zugriffszeit von Festplatten abhängen.

Auf diese Weise kann HADES die besonderen Anforderungen von eventbasierten Systemen an eine Datenbank erfüllen und die Ende-zu-Ende Laufzeit im Vergleich zu etablierten Datenbankmanagementsystemen verringern. Gleichzeitig wird durch den komplett redundanten Aufbau eine höhere Zuverlässigkeit und Verfügbarkeit erreicht.

Durch einen Wechsel weg von TCP/IP und Ethernet hin zu Netwerktechnologien, die speziell für eine Kommunikation mit niedriger Latenz entwickelt wurden, wie z.B. SCI und Myrinet und den dazugehörenden Protokoll-Stacks, bieten sich Optimierungsmöglichkeiten, die eine weitere Beschleunigung von Zugriffen um einen Faktor 10 in Aussicht stellen.

Literatur

[AvdBF+92] Peter M. G. Apers, Carel A. van den Berg, Jan Flokstra, Paul W. P. J. Grefen, Martin L. Kersten, and Annita N. Wilschut. PRISMA/DB: A Parallel Main Memory Relational DBMS. *Knowledge and Data Engineering*, 4(6):541–554, 1992.

[CRW01] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001.

[FMG03] Ludger Fiege, Gero Mühl, and Felix C. Gärtner. Modular Event-based Systems. *The Knowledge Engineering Review*, 17(4), 2003. to appear.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*, page 557 / 728. Morgan Kaufmann, 1993.

[Gär01] Felix Christoph Gärtner. *Formale Grundlagen der Fehlertoleranz in verteilten Systemen*. PhD thesis, Fachbereich Informatik, TU-Darmstadt, 2001.

[SK98] Abraham Silberschatz and Henry F. Korth. *Database System Concepts*, page 525ff / 589. Mc Graw Hill, third edition, 1998.

[WV02] Gerhard Weikum and Gottfried Vossen. *Transactional Information Systems*, page 23f. Morgan Kaufmann, 2002.