

PostgreSQL - Aktuelle Entwicklungen

7. Juni 2004

Note légale

Dieser Beitrag ist lizenziert unter der UVM Lizenz für die freie Nutzung unveränderter Inhalte.

Zusammenfassung

PostgreSQL, das fortschrittliche open-source objektrelationale Datenbankmanagementsystem, hat Freunde und Anwender in aller Welt und in allen Bereichen des Geschäftslebens. Es bietet viele Features für anspruchsvolle Anwendungen, garantiert hohe Datensicherheit und braucht sich auch bei der Geschwindigkeit nicht zu verstecken.

Der Vortrag gibt einen Überblick über die neusten Entwicklungen im Projekt und seinem Umfeld. Unter anderem wird die aktuelle Version PostgreSQL 7.4 vorgestellt, welche mehrere entscheidende Verbesserungen in den Bereichen Performance, Standard-Konformität und Benutzerfreundlichkeit enthält. Außerdem wird aus den Bereichen GUI-Werkzeuge und Hochverfügbarkeit/Replikation berichtet. Abschließend gibt es einen Ausblick auf das kommende Release und zukünftige Entwicklungen.

Dieser Vortrag richtet sich an alle Datenbank-interessierten Besucher, die sich über die aktuellen Geschehnisse im PostgreSQL-Projekt informieren möchten.

1 Einleitung

PostgreSQL <<http://www.postgresql.org/>> ist ein objektrelationales Datenbankverwaltungssystem. Es unterstützt SQL92 und SQL99 und bietet viele moderne Fähigkeiten, wie komplexe Anfragen, Fremdschlüssel, Trigger, Sichten, transaktionale Integrität und Multiversionenkontrolle. Außerdem ist PostgreSQL auf einzigartige Weise vom Benutzer erweiterbar, zum Beispiel mit Datentypen, Funktionen, Operatoren, Aggregatfunktionen, Indexmethoden oder prozeduralen Sprachen. Dieser Artikel gibt einen Überblick über die Neuerungen im Umfeld von PostgreSQL, stellt einige Features der aktuellen Release PostgreSQL 7.4 vor und gibt einen Ausblick auf zukünftige Entwicklungen. Außerdem wird noch etwas ausführlicher auf das Thema Replikation eingegangen.

2 Windows-Port

Die nächste PostgreSQL-Release wird einen nativen Windows-Port haben. Dies ist eine der größten Portierungsprojekte in der Geschichte dieser Software. Zwar ist es schon seit 1998 möglich, PostgreSQL auf Windows einzusetzen, aber geschieht dies unter Verwendung des Toolkits Cygwin, welches verschiedene Probleme mit der Performance, Benutzerfreundlichkeit und Lizenz mit sich bringt. Für Entwicklung und Experimentierung ist der Cygwin-Port durchaus geeignet, für den Produktionseinsatz aber nicht zu empfehlen.

Bereits vor zwei Jahren wurden zwei unabhängige kommerzielle Windows-Portierungen unternommen, und zwar von den Firmen SRA und PeerDirect, welche den Code kurz darauf auch der Community zur Verfügung stellten. Für eine direkte Integration war der Code leider nicht geeignet, wodurch sich die eigentlich schon länger geplante offizielle Portierung mehrmals verschob. Die Community entwickelt auf Basis dieser existierenden Erfahrungen eine Windows-Portierung, die mit dem bisherigen Code verträglicher ist und den zukünftigen Wartungsaufwand reduzieren soll.

Wichtige Probleme bei der Erstellung des Ports waren:

- PostgreSQL benutzt ein fork-Modell, aber Windows hat keinen fork-Systemaufruf. Daher musste das fork-Modell durch ein exec-Modell ergänzt werden, das heißt, die globalen Daten des Hauptprozesses

(postmaster), die von fork normalerweise automatisch an die Kindprozesse übergeben werden, müssen von Hand übertragen werden. Von einer Umwandlung des ganzen Servers in ein Thread-Modell wurde nach heftigen Diskussionen abgesehen, da damit unter anderem ladbare Erweiterungsmodule die Stabilität des ganzen Servers beeinträchtigen können würden.

- Unix-Funktionen wie link, rename, fsync sind teilweise nicht vorhanden oder haben nicht die gewünschte Wirkung. Zum Beispiel kann man unter Windows geöffnete Dateien nicht (immer / verlässlich) umbenennen. Diese Problematiken waren teilweise schon aus dem Cygwin-Port bekannt.
- Pfad-Namen, Eigenschaften von Dateinamen und Installationsschemas mussten angepasst werden.
- Ein neuer IPC-Mechanismus (Shared Memory, Semaphore) musste implementiert werden. Durch vorangegangene Portierungsarbeiten jüngerer Jahre auf andere Nicht-ganz-Unix-Betriebssysteme wie MacOS X, BeOS und QNX war hier schon eine Abstraktionsschicht geschaffen worden.
- Alle Shell-Skripte (z.B. pg_dumpall, initdb) mussten in C neu geschrieben werden. Das war erstaunlich aufwendig, weil insbesondere initdb die Möglichkeiten der Textmanipulation in der Shell, z.B. durch sed und grep, voll ausgeschöpft hatte.
- Das Unix-Signal-System steht auf Windows nicht zur Verfügung. Das Signalsystem wird von PostgreSQL ausgiebig verwendet. Mit möglichen Ersatzsystemen wird noch experimentiert.
- Die Socket-API ist unter Windows zwar vorhanden, erforderte aber trotzdem einige Anpassungen im PostgreSQL-Code.
- Das Build-System musste zum Glück nicht aufwendig angepasst werden, denn mit MinGW und MSys kann man das Build-System auf Basis von GNU Autoconf nahtlos auf Windows übernehmen. Das erfordert zwar, dass diejenigen, die PostgreSQL selbst kompilieren möchten, sich ein wenig mit den Unix-Tools vertraut machen, erspart den PostgreSQL-Entwicklern aber die Wartung eines kompletten zweiten Build-Systems. Außerdem können Anwender dadurch den freien Compiler GCC verwenden und müssen keinen kommerziellen Compiler erwerben.

Was ganz und gar abzuwarten ist, ist ob, wie von einigen gehofft, der Windows-Port die Verbreitung von PostgreSQL tatsächlich proportional zum Marktanteil dieses Betriebssystems erhöhen wird, oder ob, wie von anderen befürchtet, die Entwickler und Anwender in Zukunft mit Supportanfragen von unbedarften Windows-Anwendern überflutet werden. Sicherlich eröffnet der Windows-Port auf jeden Fall einen interessanten neuen Horizont für das PostgreSQL-Projekt.

3 Cache

Die kommende PostgreSQL-Release verwendet eine neue Cache-Replacement-Strategie ARC <http://www.almaden.ibm.com/StorageSystems/autonomic_storage/ARC/arcfast.pdf> (Adaptive Replacement Cache) als Ersatz für die bisherige Strategie LRU (Least Recently Used). Im Gegensatz zu LRU betrachtet ARC nicht nur die Zeit des letzten Zugriffs sondern auch die Häufigkeit des Zugriffs und passt sich automatisch auf eine ideale Balance zwischen diesen beiden Faktoren an. Dadurch wird auch verhindert, dass ein sequenzieller Scan oder ein Vacuum-Prozess den Cache-Inhalt zerstört. Trotzdem hat der Algorithmus wie LRU eine lineare Komplexität und verlangt im Gegensatz zu den meisten anderen Strategien keine manuelle Feinabstimmung. Die Autoren des ARC-Algorithmus erwarten eine Steigerung der Cache-Treffer-Raten in alle Anwendungsfällen gegenüber LRU und Tests der PostgreSQL-Implementierung lassen auf ähnliche positive Ergebnisse schließen.

4 Background Writer

Das Tunen von fast gleichzeitigen Schreibzugriffen von verschiedenen Sitzungen hat schon zu abenteuerlich anmutenden Konfigurationsparametern in PostgreSQL geführt. Als hoffentlich endgültige Lösung dieses Problems wird es einen Hintergrundprozess geben, der schmutzige Seiten aus dem Cache in einem Durchgang schreibt. Das reduziert die Interprozesskommunikation der PostgreSQL-Prozesse und ermöglicht bessere

Kooperation mit dem Kernel, dem PostgreSQL-Cache und dem verbesserten Vacuum. Diverse Fine-Tuning-Möglichkeiten werden auch hier gegeben sein.

5 Vacuum

Vacuum war lange Zeit eines der großen Problemthemen bei PostgreSQL. Seit Version 7.2 benötigt Vacuum allerdings keine Tabellensperren mehr und behindert daher auch die normale Datenbankaktivität nicht mehr. Verbesserungsmöglichkeiten ergeben sich vielmehr bei der Administration. Mittelfristiges Ziel ist es, Vacuum in einen mehr oder weniger automatisch nebenbei laufenden Garbage-Collection-Prozess umzuwandeln. Einen ersten Schritt in diese Richtung gab es in Version 7.4 mit dem Programm `pg_autovacuum`. Dieses Programm überwacht alle Tabellen in einem PostgreSQL-Server automatisch auf Schreibaktivitäten und löst nach einer voreingestellten Anzahl von Schreibzugriffen automatisch ein VACUUM und ein ANALYZE aus. Zur Zeit ist es ein normales Client-Programm welches im contrib-Verzeichnis des PostgreSQL-Source-Tree vertrieben wird. Aktuell wird es in den PostgreSQL-Server integriert, wo es noch bessere Konfigurations- und Analyse-Möglichkeiten haben wird und (noch) einfacher zu installieren sein wird. Allerdings ist der Einsatz auch heute schon äußerst empfehlenswert.

Darüber hinaus wird auch der Vacuum-Prozess selbst weiter optimiert. So wird man in der nächsten Release den Vacuum-Prozess «drosseln» können, das heißt er verzögert sich selbst, um den Durchsatz der normalen Datenbankaktivität weniger zu beeinträchtigen. Dies ist eine ziemlich einfache aber wirkungsvolle Optimierung. Zusammen mit dem neuen Background Writer und dem neuen ARC-Cache konnten die Antwortzeiten in einer TPC-C-Benchmark <http://developer.postgresql.org/~wieck/vacuum_cost/> mit Vacuum-Dauerbetrieb um über 80% gesenkt werden.

6 Weitere Performance-Verbesserungen

Die kontinuierliche Verbesserung der Performance ist seit jeher eines der Hauptaspekte bei der Entwicklung von PostgreSQL. Neben einigen einschneidenden strukturellen Veränderungen (z.B. MVCC, WAL, Vacuum, Background Writer) handelt es sich dabei größtenteils um Detailarbeiten, bei denen durch Profilanalysen immer neue Schwachstellen entlarvt werden, welche nach der Beseitigung wiederum durch neue Kandidaten abgelöst werden. Dabei geht es zum Teil um algorithmische Verbesserungen, teilweise aber auch um compiler-nahe Optimierungen und Tricks, wie zum Beispiel das Alignment bestimmter Speicherstellen. Denn trotzdem das Datenbanksystem in der Regel I/O-bound ist, lässt sich nach wie vor viel CPU-Zeit einsparen.

Einige Beispiele aus PostgreSQL 7.4:

- Subqueries mit IN/NOT IN wurden durch algorithmische Verbesserungen (Join-Algorithmen) um ein vielfaches beschleunigt.
- GROUP BY durch Hashs statt durch Sortieren ermöglicht dem Optimizer eine Auswahl an Aggregierungsstrategien.
- Ein komplett neues Regular-Expression-Engine wurde eingebaut.
- Inlining für einfache SQL-Funktionen spart Overhead beim Funktionsaufruf.
- Das neue Client/Server-Protokoll wurde auf bessere Startup-Zeiten getunt.
- Die Berechnung der Kostenfaktoren für Planknoten wird kontinuierlich verbessert.
- Der Datentyp numeric wurde neu implementiert und wurde dadurch erheblich performanter.

Einige Beispiele aus dem aktuellen Entwicklungsbranch:

- Das Locking im Buffer-Manager wurde überarbeitet, da es nach genauerer Analyse ein erhebliches Bottleneck bei hoher Belastung war, insbesondere auf SMP-Maschinen.
- Die intern häufig verwendete Linked-List-Implementierung (eine Tradition aus den Tagen, als die Vorgängersoftware von PostgreSQL in LISP implementiert war) wurde überarbeitet, so dass die Operationen `length` und `append` nur noch konstante Zeit beanspruchen.

- Die schon seit einiger Zeit stattfindende Evaluierung des genetischen Optimizers (GEQO) führte zu weiteren Verbesserungen.
- Weitere in der Reihe der endlosen Änderungen im Planer und Executor sparen in einigen Fällen wieder ein paar CPU-Zyklen mehr.
- Durch Performance-Profiling konnte der Spinlock-Code für Intel-CPU's verbessert werden, was in OLTP-Benchmarks bis zu 10% mehr Durchsatz brachte.

Durch immer neue Entwicklungen im Datenbankserver, Änderungen in den Betriebssystemen und neue Hardware ist davon auszugehen, dass sich dieser Tuning-Prozess unbegrenzt fortsetzen wird. Allerdings führt dies auch zu einer stetig wachsenden Zahl an Tuning-Parametern, die sich mittlerweile nur noch schwierig überblicken lassen. Eine Herausforderung für die Zukunft wird es daher sein, dem Anwender bessere Werkzeuge in die Hand zu geben um diese Tuning-Parameter zu verwalten, oder noch bessere Optimierungsmethoden zu verwenden, die sich selbst tunen können.

7 Java

Für PostgreSQL gibt es diverse Plugins, die es ermöglichen, benutzerdefinierte Funktionen in verschiedenen Programmiersprachen zu schreiben. Schon länger dabei sind PL/pgSQL, Tcl, Perl und Python; mittlerweile gibt es auch Plugins für Ruby, die Sprache R für statistische Berechnungen, und sogar einen Plugin für Shell-Skripte. Zur Zeit gibt es außerdem zwei Projekte, die an Plugins arbeiten, die das Schreiben von benutzerdefinierten Funktionen in Java ermöglichen werden.

PL/Java <<http://gborg.postgresql.org/project/pljava/projdisplay.php>> von Thomas Hallgren implementiert das bekannte Modell: der PostgreSQL-Prozess lädt den Interpreter der Sprache, in diesem Fall eine Java Virtual Machine, und führt darüber den Funktionscode aus, in diesem Fall über das Java Native Interface (JNI). Dadurch läuft der Java-Code im Serverprozess und hat direkten Zugriff auf die Strukturen des Servers. (Natürlich schützt der PL/Java-Plugin den Server vor zerstörerischen Zugriffen aus User-Code.) Unter anderem steht auch ein JDBC-Treiber zur Verfügung, mit dem man aus Java-Funktionen im Server Datenbankabfragen ausführen kann. Eine Beta-Version von PL/Java ist seit Anfang dieses Jahres verfügbar. Für die Zukunft geplant sind unter anderem die Möglichkeit, Java-Funktionen mit GCJ in Binärcode zu kompilieren und mit dem bekannten Mechanismus für C-Funktionen zu laden.

Das Projekt PL/J <<http://plj.codehaus.org/>> von Laszlo Hornyak geht einen anderen Weg: Funktionen werden über RPC auf einem speziell für diese Aufgabe laufenden Server ausgeführt. Dadurch muss nicht für jeden PostgreSQL-Serverprozess eine Java Virtual Machine gestartet werden, wodurch CPU-Zyklen und Hauptspeicher gespart werden können. Außerdem ist der PostgreSQL-Server dadurch besser gegen Abstürze der Java Virtual Machine geschützt. Theoretisch ist diese Methode sogar noch flexibler: man kann nicht nur Java mit PostgreSQL benutzen sondern möglicherweise auch andere Sprachen und andere Datenbanksysteme. Allerdings ergibt sich bei dieser Lösung natürlich ein erhöhter Kommunikationsaufwand und höhere Komplexität. PL/J ist zur Zeit noch in der Entwicklung.

Welcher dieser Ansätze sich durchsetzen wird ist zur Zeit noch völlig offen. Beide Projekte versuchen, den SQL/Java-Standard aus SQL 2003 zu befolgen, so dass eine gewisse Kompatibilität zu erwarten ist.

8 Geschachtelte Transaktionen

Mit geschachtelten Transaktionen ist es möglich, innerhalb einer Transaktion Subtransaktionen auszuführen. Damit kann man größere Aufgaben in kleinere Teilaufgaben zerlegen, die einzeln abgearbeitet werden können und dabei gegebenenfalls einzeln abgebrochen und wiederholt werden können. An geschachtelten Transaktionen für PostgreSQL wird zur Zeit gearbeitet. Ob das Feature für die nächste Release fertig wird ist zur Zeit nicht klar. Ein anspruchsvolleres Ziel von geschachtelten Transaktionen ist, dass die Subtransaktionen teilweise parallel ausgeführt werden können. Derartige Funktionalität ist zur Zeit noch nicht geplant.

9 PITR

Point-in-Time Recovery (PITR) bezieht sich auf die Fähigkeit des Datenbanksystems, beliebige Zustände in der Vergangenheit wiederherstellen zu können. Das wird normalerweise so implementiert, dass alle Datenbankaktivität in einem Log gespeichert wird, der dann sozusagen in umgekehrter Reihenfolge abgespielt werden kann um alle Aktivitäten bis zum gewünschten Punkt rückgängig zu machen. Es handelt sich dabei, um ein zweites, größeres Sicherheitsnetz, neben der Möglichkeit unerwünschte aber noch nicht abgeschlossene Transaktionen zurückrollen zu können. Außerdem lassen sich so inkrementelle Backups realisieren.

Der PostgreSQL-Server pflegt sowieso schon einen Log aller Datenbankaktivität, nämlich den Write-Ahead Log (WAL), der für das Roll Forward (Redo) im Falle eines Absturzes gehalten wird. Dieser Log lässt sich auch für das von PITR benötigte Rollback (Undo) verwenden; entsprechende kleinere Anpassungen sind bereits getätigt worden. Die restliche Arbeit, um das Feature PITR abschließen zu können, besteht hauptsächlich in der Entwicklung von Administrationswerkzeugen zum Sichern und Wiedereinspielen des Logs. Viel Wert wird insbesondere darauf gelegt, eine möglichst allgemein gehaltene Schnittstelle anzubieten, in die auch externe Backup-Software eingebunden werden kann. Diese Arbeit ist zur Zeit im Gang und wird wahrscheinlich zur neuen Release abgeschlossen sein.

10 Replikation

Replikation ist eines der meistdiskutiertesten Themen im Umfeld von PostgreSQL. Einerseits wird es von vielen Benutzern als eines der wichtigsten Features eines Datenbankmanagementsystems gesehen, andererseits kann die Entwicklergruppe bis heute keine abschließende Empfehlung und keinen definitiven Ausblick zu diesem Thema geben. In diesem Abschnitt werde ich erklären wie es zu dieser Situation gekommen ist und welche Lösungen dem Anwender heute zur Verfügung stehen.

10.1 Rückblick

Erste Nachfragen nach Replikation in PostgreSQL gab es schon 1997, stießen damals aber kaum auf Reaktionen. Damals gab es wichtigere Probleme und die Erstellung einer Replikationslösung erschien zu aufwendig. Erst als sich um die Jahre 2000/2001 kommerzielle Firmen um PostgreSQL formierten (z.B. PostgreSQL, Inc. <<http://www.pgsql.com/>> und Great Bridge LLC) kam wieder Bewegung in das Thema Replikation.

Das im Nachhinein langlebigste Produkt war eRServer <<http://www.erserver.com/>> von PostgreSQL, Inc., erstmals erschienen im Oktober 2001, eine asynchrone Master-Slave-Replikationslösung, basierend auf Triggern. Zuerst war es allerdings nur unter kommerzieller Lizenz erhältlich, wurde aber im August 2003 unter der BSD-Lizenz verfügbar gemacht. Dem Vernehmen nach werden neuere Versionen aber weiterhin hinter verschlossenen Türen entwickelt.

Ebenfalls im Jahr 2001 begann die Weiterentwicklung von Postgres-R <<http://gborg.postgresql.org/project/pgreplication/projdisplay.php>>, einem System für synchrone Master-Master-Replikation basierend auf Forschungsarbeit an der ETH Zürich. Dieses Projekt war technisch allen bis dahin verfügbaren oder angedachten Lösungen überlegen und wurde nach einem Treffen der Forschungsmannschaft mit dem PostgreSQL-Core-Team quasi als bevorzugte zukünftige Lösung auserkoren. Ende 2003 musste aber festgestellt werden, dass dieses Projekt nach zwei Jahren Entwicklungszeit zum Stillstand gekommen ist. Ein Problem war lizentechnischer Natur, ein anderes war die mangelnde Unterstützung der Entwickler sowohl moralisch als auch kommerziell. Außerdem hatten sich in der Zwischenzeit andere, wenn auch weniger mächtige Alternativen aufgetan.

Bereits im Januar 2001 wurde pgreplicator <<http://pgreplicator.sourceforge.net/>> veröffentlicht, eine asynchrone Master-Master-Replikationslösung. Dieses Produkt wurde von einer italienischen Firma entwickelt, aber noch im selben Jahr eingestellt, ohne dass sich eine große Benutzerbasis gebildet hatte.

Darüber hinaus entstanden noch eine Unmenge Insellösungen, welche letztenendes meistens spezialisierte Versionen einer Trigger-basierten asynchronen Master-Slave-Replikation waren.

Bruce Momjian lieferte Anfang 2003 auf mehreren Kongressen eine erste gute Zusammenfassung der Replikationslösungen für PostgreSQL <<http://candle.pha.pa.us/main/writings/pgsql/replication.pdf>>. Er teilte die verfügbaren bzw. denkbaren Lösungen in vier Klassen ein:

1. asynchron, Master/Slave: z.B. eRServer

2. synchron, Master/Slave (keine Implementierung vorhanden)
3. asynchron, Master/Master: z.B. pgreplicator
4. synchron, Master/Master: z.B. Postgres-R

Außerdem erwähnte er unter «andere Methoden» :

- multiported Disks
- Application-level Replication
- Data-partitioning

Auf Grund der Vielzahl von möglichen Replikationssystemen wurde frühzeitig beschlossen, dass Replikationssysteme bevorzugt nicht mit dem PostgreSQL-Code ausgeliefert werden sollen, sondern als Plugin integriert werden sollen. Durch die offene Architektur konnten die meisten Replikationslösungen (mit der großen Ausnahme Postgres-R) in der Tat als Erweiterungsmodule ohne Patch des Servers ausgeliefert werden. Obwohl diese Entscheidung sicher sinnvoll war, hat sie auch direkt zum heutigen Wildwuchs der Replikationslösungen beigetragen.

Abgesehen von eRServer hat sich keine der damals entstandenen Replikationslösungen halten oder durchsetzen können, und die Vorstellung einer einheitlichen Replikations-Plugin-API ist auch nie über die Phase einer Idee hinausgekommen.

10.2 Hochverfügbarkeit oder Replikation

Im nächsten Abschnitt werde ich eine andere Klassifizierung der Replikationslösungen vornehmen, welche die tatsächlichen Benutzerbedürfnisse und die aktuelle Marktlage besser trifft. Für Replikation gibt es prinzipiell zwei große Anwendungsgebiete: Absicherung im Fall von Hardwareschäden (Hochverfügbarkeit) und Leistungssteigerung (Load-Balancing). (Ein drittes Anwendungsgebiet ist die Verteilung der Daten auf mehrere Standorte mit nicht ausreichender Netzwerkverbindung. Eine sinnvolle Lösung dafür gibt es zur Zeit nicht.) Die PostgreSQL-Gemeinde hatte sich von Anfang an auf Load-Balancing-fähige «Replikation» konzentriert, wodurch eine Klasse von einfacheren Lösungen, welche ausschließlich Hochverfügbarkeit gewährleisten, ignoriert wurde. Allerdings stellt sich in der Praxis heraus, dass für Load-Balancing nur äußerst selten Bedarf besteht, während praktisch jeder ernsthafte Anwender sein Datenbanksystem hochverfügbar implementieren möchte.

10.3 Hochverfügbarkeitslösungen

Die verfügbaren Lösungen lassen sich wie folgt einordnen:

- redundante Hardware
- Clustering auf Betriebssystemebene
- Clustering auf Datenbankebene
- Replikation

In den folgenden Abschnitten werden ich im Einzelnen auf diese Kategorien eingehen.

10.3.1 Redundante Hardware

Hochwertige Server-Maschinen sind heute in der Regel relativ redundant ausgelegt: vom doppelten Netzteil über die hot-swappable CPUs bis zum RAID. Allerdings haben solche Maschinen alle bestimmte nicht redundante Schwachstellen, zum Beispiel den Festplatten-Controller. Daher ist eine solche Lösung nicht ausreichend.

Eine brauchbare Lösung erhält man, wenn man zwei Maschinen verbindet, entweder indem man einen Plattenstapel teilt (erhältlich von diversen Herstellern), oder indem man das Dateisystem über das Netzwerk

repliziert, zum Beispiel mit DRBD <<http://www.drbd.org/>> . Bei einem Hardwareausfall wird durch Heartbeat der aktive Rechner gewechselt. Dadurch hat man stets eine hochverfügbare Speicherung der Daten und tauscht bei Problemen einfach automatisch den davor liegenden Rechner aus. Diese Lösungen erfüllen die Anforderungen an Hochverfügbarkeit und sind mit Standard-Hardware und -Software und geringem Arbeits- und Wartungsaufwand zu realisieren. Außerdem kann man auf diese Weise auch andere Dienste, zum Beispiel Webserver oder LDAP-Server, mit nur geringem Mehraufwand absichern. Allerdings ermöglichen diese Ansätze kein Load-Balancing, weil prinzipiell immer nur eine Maschine auf die Daten zugreifen kann. Nur im Fall, dass eine Maschine sicher ausgefallen ist, darf die Ersatzmaschine den Zugriff starten.

10.3.2 Clustering auf Betriebssystemebene

Eine Erweiterung der redundanten Hardware läuft darauf hinaus, dass das Betriebssystem die Failover- und möglicherweise die Load-Balancing-Funktionalität transparent abwickelt. Anfang dieses Jahres gab die Firma Linux Labs ihr Produkt Clusgres <<http://www.linuxlabs.com/clusgres.html>> bekannt, welches diese Idee umsetzt. Clusgres ist ein geclustertes Betriebssystem, basierend auf Beowulf-Technologie, welches speziell für PostgreSQL erweitert und optimiert wurde. Es bietet Ausfallsicherheit und Load-Balancing, ist aber nicht nur für Datenbanksysteme geeignet. Es ist allerdings ein kommerzielles Produkt und erfordert spezielle Hardware. Wegen der relativen Neuheit liegen auch noch keine Erfahrungswerte vor.

10.3.3 Clustering auf Datenbankebene

Eine scheinbar populäre Realisierung von Hochverfügbarkeit und eventuell Load-Balancing ist eine ad-hoc implementierte Replikation direkt in der Anwendungssoftware. Dabei sorgt die Anwendung selbst dafür, dass Lese- und Schreibzugriffe auf mehrere Server verteilt werden. Abhängig von der Komplexität der Anwendung ist dies praktikabel und wegen der besseren Anpassungsmöglichkeiten eventuell durchaus von Vorteil. Allerdings ist eine wiederverwendbare Lösung natürlich wünschenswerter.

Die Verallgemeinerung dieses Ansatzes nenne ich Clustering auf Datenbankebene, wegen der Verwandtschaft mit dem Clustering auf Betriebssystemebene. Dabei wird ein Proxy zwischen die Clients und eine Gruppe Server geschaltet, der die Datenbankzugriffe auf mehrere Server verteilt. Lese-Zugriffe können vom Load-Balancing profitieren, Schreibzugriffe werden an alle Server weitergeleitet. Wenn ein Server ausfällt oder fehlerhaft reagiert, dann wird er aus der Clustergruppe entfernt.

Dieses System wurde in eingeschränkter Form erstmals im Jahr 2000 in Usogres <<http://usogres.good-day.net/>> von Tetsuichi Hosokawa umgesetzt. Es unterstützt nur zwei Server und geht dabei einer ganzen Reihe von Problemfällen aus dem Weg. Usogres fand damals aber wenig Beachtung und die Entwicklung ist mittlerweile eingestellt.

Ein viel versprechendes Projekt, welches die volle Komplexität dieser Architektur in Angriff genommen hat, ist C-JDBC <<http://c-jdbc.objectweb.org>> (Clustered JDBC) vom ObjectWeb-Konsortium. Diese Software greift über das JDBC-Interface auf die geclusterten Datenbanken zu und stellt diese ihrerseits als JDBC-Treiber den Anwendungen zur Verfügung. Dadurch können Änderungen in der Anwendungssoftware komplett vermieden werden. Außerdem ist diese Lösung unabhängig vom Datenbankprodukt und unterstützt sogar gemischte Gruppen. Damit der C-JDBC-Controller nicht als Achillesferse dieses Systems erscheint, kann dieser auch redundant ausgelegt werden. C-JDBC ist zur Zeit als Release Candidate 6 verfügbar.

Kurz vor Abschluss dieses Artikels wurde pgpool <<http://www.mail-archive.com/pgsql-general@postgresql.org/msg44082.html>> 1.0 von der Firma SRA veröffentlicht, welches PostgreSQL-Datenbankcluster auf Basis des PostgreSQL-Client/Server-Protokolls ermöglicht. Man hätte damit eine Alternative zu C-JDBC ohne die Bindung an Java. Eine Auswertung konnte auf Grund der knappen Zeit und der japanischen Dokumentation aber nicht mehr erfolgen.

In Zusammenhang mit derartigen Replikationsmechanismen fällt auch des öfteren das Stichwort Two-Phase-Commit (2PC). Two-Phase-Commit ist ein Protokoll, mit dem der Client (bzw. der Proxy) sicherstellen kann, dass auf mehrere Server verteilte Transaktionen alle zusammen oder gar nicht ausgeführt werden, um die Konsistenz der Mitglieder der Clustergruppe sicherzustellen. Dies ist insbesondere dann nötig, wenn auf die Gruppenmitglieder nicht ausschließlich über den Proxy zugegriffen wird, wodurch dann lokale Unterschiede entstehen können, die es abzufangen gilt. Für das 2PC-Protokoll gibt es einige standardisierte Schnittstellen und daher auch bestehende Anwendungsprogramme, die darauf aufbauen. Das Protokoll hat allerdings einige prinzipielle Fehler (unter anderem gibt es Situationen in denen es kein Vor oder Zurück gibt), wes-

wegen es eigentlich für den Produktionseinsatz nicht zu empfehlen ist. Die Erweiterung Three-Phase-Commit (3PC) beseitigt diese Fehler theoretisch, hat aber keine Unterstützung von Standards und Anwendungen. Aus diesen Gründen halten die Hauptentwickler von PostgreSQL die Implementierung von 2PC eigentlich nicht für zweckmäßig. Allerdings gibt es andere Entwickler, welche an einer Implementierung arbeiten. Anfängliche Patches liegen vor, warten aber noch auf diverse Entwicklungen bei den geschachtelten Transaktionen. Ob 2PC es in die nächste Release schafft ist daher zur Zeit äußerst fraglich.

10.3.4 Replikation

Schließlich gibt es die klassischen Replikationslösungen, bei denen zwei oder mehr Datenbankserver ihre Daten untereinander abgleichen. Eine Kategorisierung wurde bereits oben vorgenommen.

eRServer, die asynchrone Master-Slave-Replikationslösung von PostgreSQL, Inc., wurde 2003 als Open Source freigegeben. Zur Zeit ist sie die einzige freie Lösung, für die ein gewisses Maß an Erfahrungswerten vorhanden ist. Eine eRServer-Installation besteht aus einer Menge Triggern, welche Änderungen in den zu replizierenden Tabellen in eine Log-Tabelle schreiben, und einem Replikationsserver, der periodisch die geloggtten Änderungen an die Slave-Systeme verteilt. Sinnvoll ist eRServer hauptsächlich für Anwendungen, die Load-Balancing benötigen.

eRServer hat einige strukturelle Defizite: Es werden nur die Ergebnisse ganzer Transaktionen repliziert, nicht die Abfolge der Aktionen in den Transaktionen. Dadurch lassen sich bei bestimmten Constraint-Konfigurationen einige Aktionen gar nicht replizieren. Außerdem werden Schemaänderungen nicht repliziert.

Wegen dieser Probleme wurde im November 2003 von Jan Wieck das (selbst ernannte) Nachfolgeprojekt Slony-I <<http://gborg.postgresql.org/project/slony1/projdisplay.php>> ins Leben gerufen <<http://archives.postgresql.org/pgsql-general/2003-11/msg00511.php>>. Wie eRServer wird es eine asynchrone Master-Slave-Architektur. Im Gegensatz zu eRServer soll dieses System auch Schemaänderungen unterstützen und hierarchische Slave-Strukturen ermöglichen. Durch geschickte Konfiguration dieser Slave-Hierarchien kann man auch eine Art Point-in-Time Recovery (PITR) implementieren. Eine erste Beta-Version wird im Mai erwartet.

10.3.5 Zusammenfassung Replikation

Die Fokussierung auf zu hohe Ziele, die falsche Einschätzung der Anwenderbedürfnisse und das Streben des Entwicklerteams, es allen Recht machen zu wollen, führte zu einem Wildwuchs an Replikationslösungen, von denen viele unvollständig waren oder mittlerweile gescheitert sind. Trotzdem gibt es heute für Hochverfügbarkeit und Load-Balancing verschiedene praktikable und tatsächlich verfügbare Lösungen für PostgreSQL, trotzdem die hyperelegante, synchrone Multimaster-Replikationslösung weiterhin auf sich warten lässt. Für einfache Ausfallsicherung gibt es einfachere, datenbankunabhängige Lösungen. Für Load-Balancing bei Nur-Lese-Operationen bietet sich eRServer als etablierte Lösung an. Außerdem sind diverse neue und verbesserte Replikationssysteme in aktiver Entwicklung.

11 Abschlussbemerkungen

Die nächste Release, welche wahrscheinlich PostgreSQL 7.5 heißen wird, wird später in diesem Jahr erwartet. Die Beta-Phase, in der keine neuen Features mehr hinzukommen, soll im Juni oder Juli starten; die endgültige Release kann dann etwa im September oder Oktober erwartet werden. Welche der zur Zeit in Entwicklung befindlichen Features in dieser Release enthalten sein werden ist noch offen; es kommt darauf an, wie schnell die diversen noch ausstehenden Probleme gelöst werden können. Die Erfahrungen der Vergangenheit haben gezeigt, dass es alles in allem besser ist, sich an feste Daten zu halten als unbegrenzt auf die Fertigstellung bestimmter Features zu warten. Wer aktuelle Informationen zu dem einen oder anderen Feature haben möchte, der sollte die Mailingliste [pgsql-announce](http://archives.postgresql.org/pgsql-announce/) <<http://archives.postgresql.org/pgsql-announce/>> lesen, wo unter anderem wöchentliche Nachrichten aus der Entwicklung gepostet werden.

12 Danksagungen

Hans-Jürgen Schöning für spontane und ausführliche Erläuterungen des einen oder anderen Features

Dave Cramer für seine Erklärungen bezüglich des Zustandes von PL/J und PL/Java
Bruce Momjian für diverse Zusammenfassungen, Webseiten und Vorträge mit vielen Erklärungen
Allen anderen die über die Jahre offen kommuniziert und gewissenhaft dokumentiert haben, damit man
auch im Nachhinein noch darüber berichten kann