Demonstration des Parallel Data Generation Framework

Tilmann Rabl, Hatem Mousselly Sergieh, Michael Frank, Harald Kosch
Lehrstuhl für Verteilte Informationssyteme, Universität Passau
{rabl,moussell,frank,kosch}@fim.uni-passau.de

Abstract: In vielen akademischen und wirtschaftlichen Anwendungen durchbrechen die Datenmengen die Petabytegrenze. Dies stellt die Datenbankforschung vor neue Aufgaben und Forschungsfelder. Petabytes an Daten werden gewöhnlich in großen Clustern oder Clouds gespeichert. Auch wenn Clouds in den letzten Jahren sehr populär geworden sind, gibt es dennoch wenige Arbeiten zum Benchmarking von Anwendungen in Clouds. In diesem Beitrag stellen wir einen Datengenerator vor, der für die Generierung von Daten in Clouds entworfen wurde. Die Architektur des Generators ist auf einfache Erweiterbarkeit und Konfigurierbarkeit ausgelegt. Die wichtigste Eigenschaft ist die vollständige Parallelverarbeitung, die einen optimalen Speedup auf einer beliebigen Anzahl an Rechnerknoten erlaubt. Die Demonstration umfasst sowohl die Erstellung eines Schemas, als auch die Generierung mit verschiedenen Parallelisierungsgraden. Um Interessenten die Definition eigener Datenbanken zu ermöglichen, ist das Framework auch online verfügbar.

1 Einleitung

Cloudcomputing ist seit einigen Jahren ein reges Forschungsfeld. Das kontinuierliche Wachstum der Datenmengen, in vielen Anwendungen bis zu mehreren Petabytes, schafft neue Aufgaben für die Forschung. Um große Datenmengen zu verarbeiten werden automatisierte und adaptive Verfahren benötigt. In Rechnerverbunden mit tausenden von Knoten sind Hardwareausfälle keine Seltenheit, weswegen ein hohes Maß an Fehlertoleranz notwendig ist. In [RLH+09] haben wir einen Benchmark für adaptive Datenbanksysteme vorgestellt. In diesem Beitrag demonstrieren wir einen Datengenerator, der mit den Hauptzielen des Clustercomputing, Skalierbarkeit und Entkopplung, entworfen wurde.

Als Beispiel ist eine n-zu-n Beziehung zwischen zwei Relationen zu nennen. Um die Referenzen generieren zu können, müssen die existierenden Schlüssel der Relationen bekannt sein. Auf einem einzelnen Rechnerknoten ist es für gewöhnlich am schnellsten die teilnehmenden Relationen zu generieren und auszulesen um die Beziehung zu generieren. Wenn die Relationen aber über viele Rechner verteilt sind, ist es effizienter sie erneut zu generieren. Auf diese Weise kann die Beziehung vollständig unabhängig von den Basisrelationen generiert werden. Wenn für die Generierung verteilte Pseudozufallszahlengeneratoren verwendet werden, kann auch die Generation einzelner Relationen parallelisiert werden. Nachdem die Generierung deterministisch abläuft, können auch Referenzen unabhängig berechnet werden.

In [RFSK10] wurde das Parallel Data Generation Framework (PDGF) vorgestellt, das für Datengenerierung im Cloudmaßstab geeignet ist. PDGF ist hoch parallel und vollständig

konfigurierbar. Im Fokus der Implementierung standen Performanz und Erweiterbarkeit. Deshalb kann der Datengenerator auch leicht für andere Domänen eingesetzt werden. Die aktuelle Version verwendet keine Leseoperationen bei der Datengenerierung und reduziert damit die I/O und Netzwerk Last auf das absolute Minimum. Im folgenden Beitrag wird zunächst in Abschnitt 2 die Funktionsweise des Datengenerators kurz erläutert, dann in Abschnitt 3 ein Ausschnitt der Evaluierung gezeigt. Zuletzt beschreibt Abschnitt 4 die Demonstration, vor einer kurzen Zusammenfassung. PDGF ist online verfügbar[1].

2 Funktionsweise

Um einzelne Felder einer Tabelle unabhängig zu generieren, ohne teure Leseoperationen, werden jeder Spalte einer Datenbank ein Zufallszahlengenerator und ein Startwert (Seed) zugewiesen. Die verwendeten Zufallszahlengeneratoren sind hierarchisch angeordnet, wie in Abbildung 1 zu sehen. Auf diese Weise kann für jede Spalte ein deterministischer Seed erzeugt werden. Um einen einzelnen Wert in einer Spalte zu erzeugen, wird zunächst der Zufallszahlengenerator mit dem entsprechenden Seed gestartet. Dann wird die der Zeilennummer entsprechende Zufallszahl berechnet und aus dieser Zufallszahl mittels eines sogenannten Generators deterministisch der Wert erzeugt. Sollte eine einzelne Zufallszahl nicht ausreichend sein, kann die erzeugte Zufallszahl wiederum als Seed für einen Zufallszahlengenerator verwendet werden.

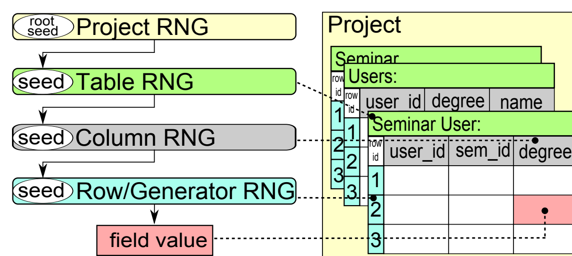

Abbildung 1: PDGFs Initialisierungsstrategie

Ein Zufallszahlengenerator wird als Startgenerator des Projekts verwendet, und dessen Startwert als Projektseed. Jeder Zufallszahlengenerator wird verwendet, um die Seeds für die in der Hierarchie nachstehenden Zufallszahlengeneratoren bzw. Generatoren zu erzeugen. So werden beispielsweise mit dem Startgenerator die Generatoren für die Tabellen geseedet. Nachdem es nur einen einzigen Projektseed gibt, können alle anderen Seeds davon abgeleitet werden und da die Anzahl der Tabellen und Spalten in einer Datenbank für gewöhnlich statisch sind, können alle Seeds im Speicher gehalten werden und müssen nur einmal bei der Initialisierung erzeugt werden. Daher muss der Datengenerator für gewöhnlich nicht die gesamte Hierarchie durchlaufen um einen Seed für einen Generator zu ermitteln. Es reicht aus den Zufallszahlengenerator erneut mit dem gespeicherten Seed zu initialisieren und zur entsprechende Zeilennummer zu springen. Danach kann die

[1]PDGF Webseite - `http://www.fim.uni-passau.de/de/home/fakultaet/lehrstuehle/verteilte-informationssysteme/forschung/dbbench.html`

Zufallszahl an den Generator weitergegeben werden.

3 Evaluierung

Um die Geschwindigkeit und Skalierbarkeit des Generators zu testen wurden TPC-H Datenbanken generiert[2]. Die Tests wurden auf einem Cluster mit 16 Knoten ausgeführt. Jeder Knoten hat zwei Intel Xeon QuadCore Prozessoren mit 2 GHz Taktfrequenz, 16 Gigabyte RAM und zwei 74 GB SATA Festplatten mit RAID 0 Konfiguration. Es wurden 2 Testreihen ausgeführt, zunächst wurde die Skalierbarkeit in Bezug auf die Datengröße getestet und danach die Skalierbarkeit in Bezug auf die Anzahl der teilnehmenden Knoten. Alle Tests zeigen, dass der Datengenerator in beiden Dimensionen linear skaliert. Die ausführlichen Testresultate können in [RFSK10] nachgelesen werden. An dieser Stelle wird nur nochmal der Vergleich mit dem in C implementierten dbgen gezeigt. Hierzu wurde mit PDGF die TPC-H Spezifizierung umgesetzt. TPC-H spezifiziert 8 Tabellen unterschiedlicher Größe und mit einer unterschiedlichen Anzahl an Spalten. Das Schema enthält Fremdschlüsselbeziehungen und verschieden Datentypen. Im Test wurde die Skalierbarkeit bezüglich der Datenmenge mit der des Standardgenerators von TPC-H - dbgen - verglichen. Dazu wurden mit beiden Generatoren Datenbanken der Größe 1, 10 und 100 GB generiert. Beide Generatoren wurden so gestartet, dass sie die 8 Kerne einer einzelnen Maschine voll ausnutzen konnten. In Abbildung 2 zeigt die Dauer der Generierung für beide Generatoren. Beide Skalen sind in logarithmischem Maßstab. Die Generierungsgeschwindigkeit beider Tools war durch die Prozessorgeschwindigkeit limitiert. Wie in der Abbildung zu erkennen ist, skaliert PDGF linear und hat eine vergleichbare Generierungsgeschwindigkeit wie eine spezialisierte C-Implementierung.

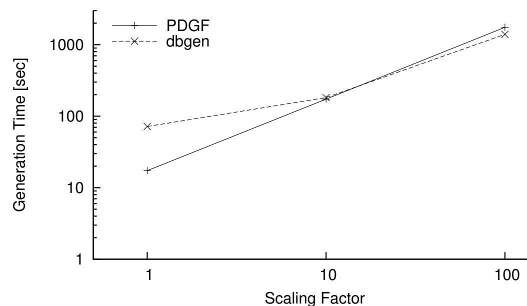

Abbildung 2: Vergleich der Generierungsgeschwindigkeit von dbgen und PDGF

4 Demonstration

Die Demonstration besteht aus zwei Teilen. Im ersten Teil wird der Datengenerator verwendet um eine einfache Datenbank zu erstellen. Im zweiten Teil wird ein einfaches “Feld-

[2]TPC-H Webseite - `http://www.tpc.org/tpch/`

Generator"-Plugin erstellt. Die Demonstration soll sowohl die einfache Anpassbarkeit, als auch Skalierbarkeit des Frameworks zeigen.

Generierung von Daten: Im ersten Teil der Demo wird ein einfaches Datenschema gezeigt und die entsprechende Datenbank generiert. Als erster Schritt wird eine XML Datei zur Schemabeschreibung erläutert und angepasst. Diese Datei enthält sowohl Parameter des Generierungsprozesses, wie die Datengröße oder das Ausgabeformat, als auch die Definition aller Schemaelemente und die entsprechenden Generierungsanweisungen. Für jede Tabelle wird aufgelistet welche Spalten existieren und entsprechende Generatoren und Verteilungen für die zu generierenden Daten spezifiziert. Nach der Spezifizierung werden Datenbanken mit verschiedenen Größen auf einer unterschiedlichen Anzahl an Rechenkernen generiert um die Skalierbarkeit zu demonstrieren.

Entwurf eines Feld-Generator-Plugin: Im zweiten Teil der Demonstration wird die einfache Erweiterbarkeit des Frameworks demonstriert, indem ein einfacher Generator erstellt wird. Die Schemadefinition wird um ein Feld, das diesen Generator verwendet erweitert und es wird erneut eine Datenbank generiert.

5 Zusammenfassung

In diesem Beitrag wurde das Parallel Data Generation Framework vorgestellt. Es ist einfach über XML Dateien anzupassen. Wie andere höher entwickelte Datengeneratoren (z.B. [SP04, HT07]) erlaubt es Abhängigkeiten zwischen Relationen und nicht-uniforme Verteilungen. Als Alleinstellungsmerkmal hat es allerdings ein neues Berechnungsmodell, das die deterministische Generierung von Zufallszahlen ausnutzt. Mit der Hilfe von Pseudozufallszahlengeneratoren können Abhängigkeiten in Datenbanken effizient aufgelöst werden, indem die referenzierten Werte erneut berechnet werden können. Die Evaluierung zeigt, dass eine generische Java-Implementierung des Modells eine äquivalente Generierungsgeschwindigkeit wie spezialisierte C-Implementierungen hat.

Literatur

[HT07] Joseph E. Hoag und Craig W. Thompson. A Parallel General-Purpose Synthetic Data Generator. *SIGMOD Record*, 36(1):19–24, 2007.

[RFSK10] Tilmann Rabl, Michael Frank, Hatem Mousselly Sergieh und Harald Kosch. A Data Generator for Cloud-Scale Benchmarking. In *TPC TC '10*, 2010.

[RLH+09] Tilmann Rabl, Andreas Lang, Thomas Hackl, Bernhard Sick und Harald Kosch. Generating Shifting Workloads to Benchmark Adaptability in Relational Database Systems. *LNCS*, 5895(2009):116–131, 2009.

[SP04] John M. Stephens und Meikel Poess. MUDD: a multi-dimensional data generator. In *WOSP '04*, Seiten 104–109, New York, NY, USA, 2004. ACM.