Performanz Evaluation von Graphdatenbanksystemen versus MySQL am Beispiel von Kookkurrenzgraphen

Martin Junghanns
martin.junghanns@studserv.uni-leipzig.de
Sascha Ludwig
s.ludwig@studserv.uni-leipzig.de
Robert Schulze
robert.schulze@studserv.uni-leipzig.de

13. Januar 2012

Zusammenfassung

Dieser Bericht ist die Zusammenfassung der Praktikumsergebnisse im Rahmen des Moduls "Fortgeschrittene Methoden des Information Retrieval" im WS 2010/11 der Universität Leipzig. Inhalt des Praktikums war die Evaluation verschiedener Graphdatenbanken und die Gegenüberstellung von relationalen Datenbanken. Als Vertreter der Graphdatenbanken wurden Neo4j, DEX und OrientDB näher betrachtet, MySQL wurde als relationale Datenbank in den Vergleich einbezogen.
Konkrete Ziele des Praktikums waren das Kennenlernen der verschiedenen Graphdatenbanken, der zu Grunde liegenden Datenmodelle und deren Abfragetechniken. Für den experimentellen Vergleich sollten verschiedene Kookkurrenzgraphen des Wortschatz Leipzig Projektes in die jeweilige Datenbanken importiert und mit den individuell angebotenen Möglichkeiten abgefragt werden. Dabei wurden ausgewählte SQL-Anfragen in API-Anfragen, aber auch in deklarative Anfragesprachen des jeweiligen Herstellers umformuliert und hinsichtlich ihrer Ausführungszeit verglichen.
Um die gemessenen Ausführungszeiten besser vergleichen zu können wurden die Messergebnisse mittels genuplot[1] visualisiert.

Todo list

1 Projektbeschreibung

Als Ergebnis des Praktikums ergaben sich zum einen konkrete und vergleichbare Messergebnisse der einzelnen Datenbanken. Zum anderen ist ein minimales, einfach konfigurierbares und auf den Anwendungsfall des Datenbankenvergleichens angepasstes Benchmarkframework entstanden. Das Framework unterstütze uns beim aggregieren der Messergebnisse einzelner Datenbanken und erleichterte wiederkehrende Aufgaben.

1.1 Softwarearchitektur

Das Framework besteht aus sechs Komponenten, durch deren Ableitung und Kombi-

[1] http://http://www.gnuplot.info/

nation sich ein einzelner Benchmarkvorgang durchführen lässt, die im folgenden kurz vorgestellt werden sollen.

1.1.1 Konnektoren

Ein Konnektor (Connector) dient den einzelnen Benchmarks oder Importierern zur Herstellung und Aufrechterhaltung einer Verbindung zur jeweiligen Datenbank. Für jede Datenbank muss nur ein Konnektor implementiert werden.

1.1.2 Importierer

Ein Importierer (Importer) dient dem Importiervorgang einer Datenbasis in die jeweilige Datenbank. In unserem Fall wurde die Datenbasis aus einer MySQL Datenbank übernommen. An dieser Stelle könnte aber auch ein Import aus einer CSV-Datei oder mit einer anderen Quelle implementiert werden. Für den vorstellbaren Fall, dass Benchmarks nicht auf einer existierenden Datenbasis operieren, ist die Implementierung eines Importierers nicht notwendig. Durch den Importierer lässt sich eine Datenbank auch in den Urzustand zurücksetzen und alle enthaltenen Daten entfernen.

1.1.3 Benchmark-Suite

Eine Benchmark-Suite dient der Zusammenfassung mehrerer einzelner Benchmark zu einer Folge, die geordnet durchgeführt werden sollen. Die Benchmark-Suite ermöglicht das Aggregieren und Persistieren einzelner Messergebnisse. In der Benchmark-Suite kann konfiguriert werden ob der eigentlichen Messung vorausgehend eine sogenannte Aufwärmung der Datenbank durchgeführt werden soll oder nicht.

1.1.4 Benchmark

Ein Benchmark misst die durchschnittliche Ausführungszeit einer Datenbankoperation. Die auszuführende Datenbankoperation ist an dieser Stelle zu implementieren.

1.1.5 Konfigurationsdateien

Bezüglich geeigneter Parameter ist die Konfiguration der einzelnen Datenbanken oder des Frameworks selbst über sogenannte properties-Dateien möglich.

1.1.6 Auswärtungsskripte

1.2 Bedienungsanleitung

1.2.1 Vorraussetzungen

- repository auschecken
- maven dependencies installieren
- etc.
- pp.

1.2.2 Benchmarks ausführen

1.2.3 Benchmarks konfigurieren

1.2.4 zusätzliche Benchmarks

1.2.5 zusätzliche Datenbank

1.3 Einschränkungen

Vorhandene Einschränkungen Datenbank-spezifischer Natur sind im jeweiligen Abschnitt dokumentiert.

An dieser Stelle sei aber auch erwähnt, dass wir im Rahmen des Praktikums keine überdurchschnittliche Expertise bezüglich der verwendeten Datenbanken ausbilden konnten und so die Aussagekraft unserer Messergebnisse unter folgendem Bedingungen zu betrachten sind: Es ist durchaus möglich, dass sich die in den Benchmarks verwendeten Datenbankoperationen noch weiter optimieren ließen. Dabei kann es sein, dass eine Optimierung nur hinsichtlich bestimmter Operationen einen Vorteil bringt und sich bei andere Operationen eher nachteilig auswirkt.

1.4 Erweiterungsmöglichkeiten

Bezüglich der Erweiterungsmöglichkeiten muss an dieser Stelle unterschieden werden. Zum einen ist es möglich die Benchmarks an sich zu erweitern, dazu siehe die Bedienungsanleitung im Absatz 1.2. Zum anderen bietet das Framework an sich Potenzial für Erweiterungen.

Vereinfachte Konfiguration der Benchmark-Suiten

Es ist vorstellbar, dass die Erstellung einer Benchmark-Suite, also das Zusammenführen einzelner Benchmarks zu einer geordneten Folge, nicht zwingend in Programmierleistung in einer Java-Klasse geschehen muss. Denkbar wäre, dass man dies lediglich in einer properties-Datei konfiguriert. Dies vereinfacht das Anpassen einer Benchmark-Suite und ist auch für Personen ohne Programmierkenntnisse zugänglich.

Visualisierung von Messergebnissen durch das Framework

Momentan geschieht die Auswertung der Messergebnisse noch außerhalb des Frameworks. Dazu existiert ein zusätzliches python-Skript, welches die Messergebnisse aggregiert, aufbereitet und mittels gnuplot visuell zugänglich ausgibt. Dieses aggregieren, aufbereiten und Ansprechen von gnuplot ist auch in Java möglich und könnte in das Framework integriert werden.

2 Neo4j

Neo4j[2] ist eine eingebettete, persistente, transaktionale Graphdatenbank. Sie wird seit 2005 von der Firma Neo Technology, Inc. entwickelt, Version 1.0 erschien Anfang 2010. Die Datenbank wurde vollständig in Java geschrieben und steht quelloffen zur Verfügung.[3]

2.1 Anbindung

Neo4j kann in zwei verschiedenen Szenarien betrieben werden: Die eingebettete Version wird direkt in eine Java Applikation integriert und bietet den performantesten Zugriff auf die Datenbank. Darüber hinaus implementiert Neo4j eine REST Schnittstelle, mit welcher auf die Datenbank via HTTP Requests zugegriffen werden kann. Die Anbindung erfolgt über verschiedene Clients, welche den Zugang aus unterschiedlichen Plattformen und Sprachen heraus ermöglichen.
Für die Leistungsmessung wurde die eingebettete Variante der Datenbank genutzt. Sie bietet neben dem Geschwindigkeitsvorteil auch das volle Spektrum an Zugriffsmöglichkeiten an.

2.2 Anfragemöglichkeiten

Neo4j bietet verschiedene Möglichkeiten an, den gespeicherten Graphen zu traversieren

[2] `http://www.neo4j.org`
[3] `https://github.com/neo4j/community`

bzw. abzufragen. Neben der nativen Java API und der ebenfalls in Java entwickelten Traversal API steht seit Version 1.5 mit Cypher eine deklarative Abfragesprache zur Verfügung. Die drei genannten Abfragemöglichkeiten werden im Folgenden kurz vorgestellt und die Formulierung der vorgegebenen SQL Queries aufgezeigt.

2.2.1 Native API

2.2.2 Traversal API

2.2.3 Cypher

2.3 Einschränkungen

Cypher, Traverser Speicherprobs, keine Optimierung

2.4 Dokumentation

2.5 Aussicht

3 OrientDB

3.1 Anfragemöglichkeiten

OrientDB lässt sich einfach über eine Java API steuern. Das Handling von OrientDB wird, nach einer gewissen Einarbeitungszeit, relativ einfach, was auch der guten Javadoc der API geschuldet ist. Neben einer Java-API bietet OrientDB auch die Möglichkeit der Steuerung des Datenbanksystems über eine Konsole. Dies hat den Vorteil, dass man das Datenbanksystem erkunden und kennen lernen kann. In der Konsolen können Befehle in einer SQL-artigen Syntax an das DBS geschickt und ausgewertet werden.

3.2 Einschränkungen

Ein großes Bottleneck bei OrientDB ist der Import der Daten aus einer SQL-Datenbank. Auch nach dem befolgen der "Performance Hintsünd dem Optimieren diverser JVM Parameter, stieg die Zeit für den Import exponentiell zur Anzahl der einzufügenden Daten an. Dies hat zur Folge, dass große Datenbanken (ab 1M Datensatz) nicht mehr importiert werden konnten. Mittels JProfiler konnte der Persistierungsprozess von OrientDB als HotSpot ausgemacht werden. Ein Kontaktaufnahme mit dem Autor bezüglich dieses Performanzverlustes ist durch uns erfolgt.

3.3 Dokumentation

OrientDB ist gut und ausführlich dokumentiert. Es liegt sowohl eine komplette und ausführliche Javadoc der API vor, als auch Informationsseiten zu grundlegenden und weiterführenden Aspekten vor. Auf diesen Webseiten lassen sich Informationen zu fast allen Fragen finden. Allerdings ist das Finden einer Antwort auf eine konkrete Frage teilweise schwierig. Dies ist unter Umständen dem Fakt geschuldet, dass OrientDB sowohl ein Dokumenten-Datenbanksystem, als auch ein Key-Value-Store, als auch ein Graphdatenbanksystem ist. Oft werden in der Dokumentation diese drei Konzepte vermischt, was zu Verwirrungen führen kann. Weiterhin hat das Fehlen eines ausführlichen und kompletten Einstiegbeispiels die Einarbeitung erschwert. Auch fehlen Beispiele für komplexe Anwendungsfälle. Allerdings war der Autor bei speziellen Fragen per E-Mail zu erreichen, wobei man auf eine Antwort lediglich eine angemessene Zeit warten musste.

3.4 Aussicht

OrientDB stellt bezüglich seiner Graphdatenbankenfunktionalität einige interssante Features in Aussicht. OrientDB stellt seit Kurzem eine Implementation der Tinkerpop Blueprints APIs zur Verfügung. Somit steht auch die einfache und leicht lesbare Highlevel-Abfragesprache Gremlin zur verfügbar. Mit dieser Abfragesprache ist es möglich auch komplexe und vielseitige Abfragen verständlich zu formulieren.

4 DEX

5 Evaluation

5.1 Testbed

Hardware Fakten Datensätze (Anzahl Knoten, Kanten, evtl. Gradverteilung) cold caches / warm caches JVM settings versionen

5.2 Ergebnisse