Exposé zur Diplomarbeit

"Kostenmodelle zur Beantwortung von SparQL-Anfragen mittels RDFMatView"

Hagen Möbius
hagen.moebius@googlemail.com
180056

Betreuer

Prof. Ulf Leser
Roger Castillo

Hintergrund

RDF [7] ist ein Standard Modell des W3C, welches die Intention hat, eine universelle Möglichkeit zum Austausch von semantischen Informationen zu bieten. Jede einzelne Information wird in Form eines Tripels kodiert: (Subjekt, Prädikat, Objekt). Solch ein Tripel setzt zwei Dinge, das Subjekt und das Objekt, über das Prädikat in Beziehung zueinander. Eine Menge derartiger Tripel bildet eine Datenbasis und kann als ein gerichteter Graph dargestellt werden, in dem Subjekte und Objekte durch Knoten repräsentiert werden und Prädikate durch Kanten zwischen diesen.

Um Wissen aus einer RDF Datenbasis zu extrahieren gibt es diverse Anfragesprachen, wovon SparQL [4] vom W3C als Empfehlung ausgerufen wurde. Analog zu SQL ist SparQL eine deklarative Sprache, mittels derer man die inhaltlichen und strukturellen Bedingungen formuliert, die man an das Ergebnis stellt. Das geschieht unter Verwendung einer Menge von Tripelmustern, in denen Subjekte, Prädikate und Objekte durch Variable vertreten werden können. Nachdem die Anfrage von einer entsprechenden Engine bearbeitet und ausgeführt wurde, erhält man alle Vorkommen des Anfragemusters in der RDF Datenbasis.

Auch Anfragemuster lassen sich als Graph darstellen. Die Suche nach einem Vorkommen des Anfragemusters ist gleichzusetzen mit der Suche nach allen Vorkommen des inhaltlich und strukturell eingeschränkten Teilgraph im gesamten RDF Graph.

Die Beantwortung von Anfragen nimmt Zeit in Anspruch, die im Wesentliche von zwei Faktoren abhängt. Zum einen von der Größe der Datenbasis, also der Anzahl der darin gespeicherten Informationstripel. RDF Datenbanken werden derzeit oftmals in relationalen Datenbanken gespeichert (Sesam [5], Jena [6]), wobei häufig jedes Informationstripel als eine Tabellenzeile innerhalb einer großen Tripeltabelle repräsentiert wird. Zum anderen hängt die tatsächlich beanspruchte Zeit von der Komplexität der Anfrage ab, also von der Anzahl, der darin enthaltenen Tripelmuster. In den Anfrageengines für RDF Datenbanken,die auf einer Tripeltabelle basieren, wird eine Anfrage mit n Tripelmustern mittels eines (n - 1)-Way Self-Joins auf der großen Tripeltabelle bearbeitet. Die Zeit ist also prinzipiell exponentiell abhängig von der Anzahl der Tripelmuster. In großen RDF Datenbanken mit mehreren Millionen gespeicherten Tripeln ist dieses Lauzeitverhalten absolut unpraktikabel. Es sind Ansätze und Algorithmen zur Beschleunigung der Bearbeitung gefragt. Einer dieser Ansätze ist RDFMatView. [2]

RDFMatView

Die Idee von RDFMatView ist, eine Anfrage P_I auf der Datenbasis auszuführen und deren Ergebnisse O_I als so genannten materialisierten Index $I = (P_I, O_I)$ zu speichern. Dieser Index beinhaltet also alle Vorkommen des Indexmusters und bildet damit einen persistenten Cache, welcher die durch das Indexmuster erforderten Self-Joins über der Datenbasis abkürzt. Wann immer eine darauf folgende Anfrage Q mit dem Muster P_Q, ein Vorkommen des Indexmusters enthält, kann diese Anfrage umgeschrieben werden. Die im Index enthaltenen Self-Joins können damit gespart werden. Allerdings müssen die durch den Index exponierten

Vorkommen O_I noch durch die restlichen Bedingungen $P_Q \setminus P_I$ erweitert werden. Eine Anfrage wird unter Verwendung eines Indexes also im Allgemeinen beschleunigt. Ist das Anfragemuster gleich dem Indexmuster, so ist das Ergebnis O_Q der Anfrage, gleich den Indexvorkommen O_I, weil dann der Index das Ergebnis vollständig vorwegnimmt.

RDFMatView kann aber nicht nur, wie eben beschrieben, mit einem einzelnen Index arbeiten, sondern erlaubt das Anlegen einer Menge von Indexen, sowie die Verwendung mehrerer Indexe zu Beantwortung einer Anfrage. Bei der Bearbeitung einer Anfrage stellt sich dann für jeden angelegten Index die Frage, ob er für die Anfrage zulässig ist. Dies bildet die Menge aller zulässigen Indexe und prinzipiell kann jede Teilmenge davon verwendet werden, um die Anfrage um zuschreiben und damit eventuell zu beschleunigen. Allerdings produziert jede Teilmenge natürlich einen anderen Zeitgewinn. Das liegt einerseits an festen Eigenschaften der einzelnen Indexe und andererseits an emergenten Eigenschaften bei konkreten Kombinationen von Indexen.

Ziel

Ziel der Arbeit ist es, eine verbesserte Kostenfunktion zu entwickeln, welche die Auswahl der optimalen Indexkombination ermöglicht. Dazu wird ein heuristisches Modell benötigt, das die Eigenschaften der einzelnen Indexe, Eigenschaften von Kombinationen von Indexen sowie generelle Statistiken der Datenbasis geeignet miteinander kombiniert.

Die Kostenfunktion soll zu einer Anfrage Q und einer konkreten Überdeckung $C = \{I_1, \ldots, I_n\}$ eine Abschätzung der Kosten - also des zu erwartenden Zeitaufwandes - liefern. Durch diesen Skalarwert wird es dann möglich, die Menge der Überdeckungen nach ihrem Zeitaufwand zu ordnen und die optimale Lösung auszusuchen.

Dabei ist es wünschenswert, dass das Ergebnis der Kostenfunktion proportional zum tatsächlichen Zeitaufwand ist, da in diesem Fall begründet behauptet werden kann, dass die Funktion auch im Allgemeinen sinnvolle Ergebnisse liefert. Im Prinzip ist es hinreichend, der Kombination mit den tatsächlich geringsten Kosten den niedrigsten Wert zu geben - notwendigerweise sollen aber besonders schlechte Indexkombinationen vermieden werden.

Es gibt drei Maßstäbe für die zu entwickelnde Kostenfunktion: Güte, Zeit und Platz. Die Güte der Funktion soll natürlich so hoch wie möglich sein; dabei sollen die Zeitkosten zu ihrer Berechnung und der Platzbedarf für die Statistiken, die sie dafür verwendet, so gering wie möglich sein. Es gilt aber im Allgemeinen, dass mit steigender Güte sowohl Zeit als auch Platz steigen. Gute Schätzungen benötigen gute Statistiken und diese verbrauchen entweder viel Platz oder viel Zeit, sollten sie online berechnet werden. Andererseits muss man sich für weniger Platz- und Zeitverbrauch meist mit geringerer Güte zufrieden geben. Somit stehen die drei Maßzahlen zueinander in Konkurenz und ein weiteres Ziel der Arbeit ist, ein zufriedenstellendes "Gleichgewicht" für diesen Trade-Off zu finden.

Die Kostenfunktion darf natürlich nicht so komplex sein, dass jeglicher Zeitgewinn ausgelöscht wird, der durch die Verwendung der ermittelten optimalen Überdeckung entsteht. Die Berechnung muss auf Werten basieren, die offline berechnet werden können. Es bieten

sich Statistiken und Maßzahlen an, die zum Zeitpunkt der Indexierung berechnet werden können.

Bisherige Arbeiten

Bei RDFMatView wurde eine Maßzahl für einzelne Indexe ermittelt: die Selektivität, die auf den zwei folgenden Grundsätzen basiert. Angenommen, wir haben für eine Anfrage Q zwei zulässige Indexe I_1 und I_2 ermittelt. Welcher von beiden den größeren Zeitgewinn erzeugt, hängt von zwei fundamentalen Eigenschaften der Indexe ab.

1. Der Index, der aus der größeren Anzahl von Tripelmustern erzeugt wurde.
    - Eine größere Anzahl von Tripeln im Indexmuster bedeuten eine größere Anzahl von gesparten Self-Joins. Da der Index zulässig ist, wird dadurch eine größere Anzahl von Tripelmustern der Anfrage abgedeckt. Das heißt wiederum, dass die Vorkommen des Indexes um weniger Tripelmuster aus dem Rest der Anfrage erweitert werden müssen.
2. Der Index, der weniger Vorkommen in der Datenbasis hat.
    - Alle Vorkommen eines Indexes müssen durch die nicht abgedeckten Bedingungen der Anfrage erweitert werden. Hat ein Index weniger Vorkommen in der Datenbasis, so verringert sich die Anzahl Joins zwischen den überdeckenden Indexen, sowie die Anzahl der durchzuführenden Erweiterungen um Tripelmuster, die nicht durch Indexe abgedeckt sind.

Diese beiden Aussagen führen zur Definition der Selektivität eines Indexes. Je geringer die Selektivität ist, desto besser ist der Index geeignet um Anfragen möglichst schnell zu beantworten. Der Divisor spiegelt die erste Aussage wieder, indem mehr Tripelmuster den Exponent und damit den Divisor vergrößert. Die zweite Aussage wird im Dividend abgebildet.

$$sel(I) = \frac{\#(I)}{|G|^{|I|}}$$

Dieser Ansatz wurde erweitert, um die Selektivität einer Menge von Indexen berechnen zu können. Bei dieser Abschätzung wird die Selektivität im Allgemeinen größer und damit schlechter. Das liegt daran, dass die Vorkommen der Indexe miteinander kombiniert werden müssen um alle Ergebnisse für die Anfrage zu erhalten. Dieser Umstand bedingt das Produkt im Dividend. Der Divisor ist eine grobe untere Abschätzung für die Größe des kombinierten Musters der Indexe. Fasst man die beteiligten Indexmuster in einem Muster zusammen, dann muss dieses mindestens so groß sein wie das größte Einzelmuster.

$$sel(I_1, ..., I_n) \leq \frac{\prod_{i=1}^{n} \#(I_i)}{|G|^{max(|I_1|,...,|I_n|)}}$$

Herangehensweise

In [3] wird eine Evaluation der bisherigen Kostenfunktion vorgenommen. Teilweise sind die Ergebnisse zufriedenstellend, jedoch zeigt sich auch erratisches Verhalten. Desweiteren erscheint die Auflösung der Kostenfunktion zu grob, da sie in einem der beiden Beispiele für viele Überdeckungen nur zwei verschiedene Kosten errechnen kann, obwohl die tatsächlichen Kosten erhebliche Unterschiede aufweisen.

Das liegt an der Granularität der Abschätzung. Wie im vorigen Abschnitt gesehen, wird die Selektivitätsabschätzung für Kombinationen von Indexen auf die grundlegenden Eigenschaften der beteiligten Indexe zurückgeführt: die Größe (|I|) und die Häufigkeit (#(I)). Die Abschätzung geht also nicht auf die innere Struktur der Indexe ein. Insbesondere werden die Frequenzen der einzelnen Tripelmuster vernachlässigt, aus denen die Indexe bestehen, sowie die Abhängigkeiten der Indexe untereinander.

Die Diplomarbeit soll Möglichkeiten ergründen, die Indexe auf ihrer konstituierenden Ebene zu betrachten, sowie Chancen, die sich aus diesem Ansatz ergeben. Die in den Indexen enthaltenen Tripelmuster sollen einzeln auf ihre Selektivität hin analysiert werden. Zum Zeitpunkt der Indexierung werden also zusätzliche Berechnungen durchgeführt und die Ergebnisse als eine weitere Art Statistik in der Datenbank gespeichert werden.

Zum Thema der Anbhängikeiten der Indexe untereinander, enthält der Report [3] noch eine weitere Abschätzung. Für den Fall, dass die Indexe alle intensional überlappen kann eine strengere Abschätzung getroffen werden, als die oben genannte. Allerdings wird auch hierbei nicht die innere Struktur der Indexe betrachtet. Es wird lediglich vorausgesetzt, dass die Indexe überlappen, jedoch diese Überlappung nicht quantifiziert. Die Größe der Überlappung wird nicht betrachtet; auch nicht statistische Merkmale, wie zum Beispiel Kardinalitäten der überlappenden Teile.

Auch in diesem Fall ergeben sich weitere Informationen, die zum Zeitpunkt der Indexierung ermittelt werden können und die helfen können, die optimale Überdeckung zu finden. Die Kostenfunktion muss diese zusätzlichen Informationen dazu geeignet miteinander kombinieren um eine bessere Abschätzung für die tatsächlichen Kosten der Anfragebearbeitung treffen zu können.

Drei Punkte werden bei der Bearbeitung der Arbeit von Interesse sein, wobei der erste vorläufig als der Schwerpunkt gilt:

- Berechnung und Bereitstellung der Selektivitäten einzelner Tripelmuster, die in Indexen verwendet werden.
- Qualifizierung und Quantifizieren der Abhängigkeiten von Indexe untereinander. Zu diesem Thema hat die Diplomarbeit von Christian Rothe einige Ansätze, die noch nicht in der derzeitigen Implementation enthalten sind und nachgeholt werden sollten, sofern sich die Relevanz bestätigt. (Kerne von Überlappungen sowie Verzweigungsgrad eines Indexes)

- Statistische Qualifizierung der Tripelmuster aus der Anfrage, die nicht durch Indexe überdeckt sind.

Die entwickelte Kostenfunktion soll dann ausgiebigen Test anhand einer exemplarischen Anfragenmenge ausgesetzt werden. Die Anfragen und Indexe sollen die heraus gearbeiteten Stärken und Schwächen der bisherigen Kostenfunktion exploitieren und dem entwickelten Modell gegenüberstellen, um Verbesserungen deutlich zu machen.

Referenzen

[1] Thomas Neumann and Gerhard Weikum: RDF-3X: a RISC-style engine for RDF (*Proc. VLDB Endow. 1, 1, August 2008, S. 647-659*)

[2] Christian Rothe: Indexierung von RDF-Daten für SparQL-Anfragen (*Diplomarbeit, 2007*)

[3] Roger Castillo, Christian Rothe and Ulf Leser: RDFMatView: Indexing RDF Data for SparQL Queries (*Technical Report, 2010*)

[4] Eric Prud'Hommeaux, Andy Seaborne: SPARQL Query Language for RDF (*World Wide Web Consortium, Recommendation REC-rdf-sparql-query-20080115, January 2008*)

[5] Jeen Broekstra, Arjohn Kampman, Frank van Harmelen: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema (*Proc. Int'l Semantic Web Conference 2008, Vol. 2342, S. 54-68*)

[6] J. J. Carroll, I. Dickinson, C. Dollin, D.Reynolds, A. Seaborne, and K. Wilkinson: Jena: implementing the semantic web recommendations *(Proc. 13th World Wide Web Conference on Alternate Track Papers & Posters 2004, S.* 74-83)

[7] Frank Manola and Eric Miller: RDF Primer (*W3C Recommendation, February 2004*)