Das View Selection Problem

Benjamin Daeumlich

daeumlic@informatik.hu-berlin.de

Zusammenfassung: Das View Selection Problem ist das Problem der Auswahl einer Menge von zu materialisierenden Sichten (Views), so dass die Kosten der Abarbeitung von gegebenen Anfragen (Queries) möglichst gering wird und außerdem eine gegebene Beschränkung im Speicherplatz eingehalten wird. Dieses Dokument gibt nach einer kurzen Einführung in die Problematik eine formale Definition des Problems, betrachtet das View Selection Problem anschließend unter zwei verschiedenen Annahmen und gibt letztlich noch einen Lösungsansatz.

1 Einführung

Dieser Abschnitt gibt eine Motivation für das View Selection Problem. Anschließend wird anhand eines Beispiels angedeutet, dass das Problem sehr komplex ist.

1.1 Motivation

Es hat sich herausgestellt, dass materialisierte Views, welche im Gegensatz zu klassischen Views keine virtuellen sondern reale Tabellen sind, hervorragend zur Beschleunigung der Queryausführung geeignet sind. Das Ziel des View Selection Prozesses ist es eine Menge von materialisierten Views zu finden, welche die Kosten der Ausführung von mehreren Queries möglichst gering halten. Des weiteren soll dabei ein gewisser Speicherplatz nicht überschritten werden.
Neben der Anwendung für die Beschleunigung der Queryausführung in normalen Datenbankmanagementsystemen findet es auch Anwendung im Data-Warehousing-Bereich, aus welchem ursprünglich die Motivation für das View Selection Problem kam. Dort muss beispielsweise entschieden werden, welche Views im Data-Warehouse gespeichert werden sollen um eine optimale Performanz zu erhalten.
Nun stellt sich die Frage, welche Views überhaupt zum Materialiseren in Frage kommen. Ein intuitiver Ansatz wäre es davon auszugehen, dass die Views stets Teilausdrücke der gegebenen Queries sein müssen. Das bedeutet, dass die Views nur Relationen und Joinprädikate beinhalten, welche auch in den Queries vorkommen. Dieser intuitive Ansatz wird im folgenden Beispiel aus [CHS01] allerdings widerlegt.

1.2 Beispiel

Man betrachte ein Logistikunternehmen, welches verschiedene Städte beliefert. Dabei existieren festgelegte Lieferpläne zwischen Paaren von Städten. Das Unternehmen verwaltet eine Datenbank, welche eine große Tabelle `T(von,tag,nach)` beinhaltet, in der alle Lieferpläne zwischen den beiden Städten `von` und `nach` inklusive dem Liefertag `tag` (eine Zahl zwischen 1 und 7) gespeichert werden. Nun möchte das Unternehmen an die Fahrer Touren aushändigen, welche stets in der selben Stadt starten und enden. Diese Touren werden über Anfragen an die Datenbank ermittelt. Man betrachte nun die folgenden zwei Touren inklusive der zugehörigen Queries in Datalog-Notation:

- Tour1: „Eine Stadt in zwei Tagen“

```
Q1(X1) :- T(X1,D,X2), T(X2,D+1,X1).
```

- Tour2: „Fünf Städte in fünf Tagen mit Pause“

```
Q2(X1) :- T(X1,1,X2), T(X2,1,X3), T(X3,2,X4),
          T(X4,2,X5), T(X5,4,X4), T(X4,5,X1).
```

Nun stelle man sich vor, dass das Unternehmen sehr viele solcher Touren definieren will, welche alle zwischen 2 und 10 Städte beinhalten sollen. Daraus ergibt sich dann eine Reihe von Queries $\mathcal{Q} = \mathcal{Q}_1, \mathcal{Q}_2, ...$ Wie kann man die Abarbeitung dieser Queries beschleunigen? Man kann alle Queries im Voraus berechnen, allerdings wird dazu der Speicher nicht ausreichen. Weiterhin kann man sich vorstellen, eine View zu materialisieren, welche die Menge der Städte die einen Zyklus bilden beinhaltet. Für Zyklen der Länge 5 würde sich die folgende View ergeben:

- View1:

```
C5(X1) :- T(X1,D1,X2), T(X2,D2,X3), T(X3,D3,X4),
          T(X4,D4,X5), T(X5,D5,X1).
```

Offensichtlich lässt sich diese View dazu benutzen das Query `Q2` zu beschleunigen, da alle Antworten auf `Q2` in der View `C5` enthalten sind. Allerdings kann `Q1` nicht beschleunigt werden, da keine Beziehung zwischen Zyklen der Länge 2 und 5 exisitert. Somit müsste man zur Beschleunigung aller Queries in $\mathcal{Q}$ die Zyklen `C2`,`C3`,...,`C10` matierialisieren, allerdings ist dies wiederum sehr speicheraufwendig.
Eine weitere Idee ist es, eine einzige View zu materialiseren, mit der man eine Kette von Städten berechnen kann (z.B. der Länge 10):

- View2:

```
V10(X1) :- T(X1,D1,X2), T(X2,D2,X3),
           ... , T(X9,D9,X10).
```

Man kann sich nun leicht überlegen, dass alle Zyklen die in ihrer Länge kleiner oder gleich 10 sind in dieser View enthalten sind, da eine Kette von Städten auch die entsprechenden Zyklen „umwickelt“. Somit kann man `Q1` und `Q2` mittels `V10` wie folgt umschreiben:

```
Q1(X1) :- V10(X1), T(X1,D,X2), V10(X2),
          T(X2,D+1,X1).

Q2(X1) :- V10(X1), T(X1,1,X2), T(X2,1,X3), T(X3,2,X4),
          V10(X4), T(X4,2,X5), T(X5,4,X4), T(X4,5,X1).
```

Um zu zeigen, dass dies eine Beschleunigung darstellt, betrachte man den Ausführungsplan $(V_{10} \rhd\lhd T) \rhd\lhd (V_{10} \rhd\lhd T)$ für `Q1`. Diese Ausführung ist effizienter als $T \rhd\lhd T$, da das Zwischenergebnis $V_{10} \rhd\lhd T$ kleiner ist als T.

Dadurch wurde gezeigt, dass die zu materialiserenden Views nicht zwangsläufig Teilausdrücke der gegebenen Queries sein müssen. Somit wird der Suchraum extrem groß und das Problem sehr komplex.

2 Formale Definition

In diesem Abschnitt wird das View Selection Problem nach einigen Vorbetrachtungen formal definiert.

2.1 Vorbetrachtungen

Vor der formalen Definition des View Selection Problem müssen einige Begriffe definiert werden, welche man für die Definition benötigt. Dies sind zum einen die Begriffe des Workload und der Viewkonfiguration und zum anderen ist es der Begriff der Kostenfunktion.

Definition 2.1 (Workload) *Ein Workload ist eine Abfolge von Queries* $\mathcal{Q} = \mathcal{Q}_1, \mathcal{Q}_2, ..., \mathcal{Q}_m$*, wobei jedem Query* $\mathcal{Q}_i$ *ein Gewicht* ω_i *zugeordnet wird.*

Es gilt: $\sum_{i=1}^{m} \omega_i = 1$

Definition 2.2 (Viewkonfiguration) *Eine Viewkonfiguration* $\mathcal{V}$ *ist eine Menge von Views.*

Um verschiedene Views bzw. Viewkonfigurationen bewerten zu können, werden Kostenfunktionen benötigt. Es existeren drei verschiedene Funktionen welche betrachtet werden:

- $\mathcal{C}(\mathcal{R}, \mathcal{V}, \mathcal{Q}_i)$: Gibt die Kosten für die Ausführung von Query $\mathcal{Q}_i$ bei gegebenem Schema $\mathcal{R}$ und gegebener Viewkonfiguration $\mathcal{V}$ an. Diese Funktion benutzt zur Abschätzung die durch E approximierten Größen.

- $E(\mathcal{R}, V)$: Berechnet die Größe einer View V basierend auf den zu $\mathcal{R}$ gehörenden Statistiken $\Sigma_{\mathcal{R}}$. Die Statistiken beinhalten beispielsweise Selektivitätsfaktoren oder Kardinalitäten der Relationen.
- $\mathcal{C}(\mathcal{R}, \mathcal{V}, \mathcal{Q}) = \sum_{i=1}^{m} \mathcal{C}(\mathcal{R}, \mathcal{V}, \mathcal{Q}_i) \times \omega_i$: Gibt die Gesamtkosten für die Ausführung des Workload $\mathcal{Q}$ bei gegebenem Schema $\mathcal{R}$ und gegebener Viewkonfiguation $\mathcal{V}$ an.

2.2 Definition

Durch die Vorbetrachtungen lässt sich das View Selection Problem nun formal definieren.

Definition 2.3 (View Selection Problem) *Sei $\mathcal{R}$ ein Datenbankschema, $\mathcal{B}$ der verfügbare Speicherplatz, $\mathcal{Q}$ ein Workload und $\mathcal{C}(\mathcal{R}, \mathcal{V}, \mathcal{Q})$ eine Kostenabschätzungsfunktion.*

Das Problem, eine auf $\mathcal{R}$ definierte Viewkonfiguration $\mathcal{V}$ zu finden, deren Größe durch $\mathcal{B}$ beschränkt ist und die die Kostenfunktion $\mathcal{C}(\mathcal{R}, \mathcal{V}, \mathcal{Q})$ minimiert, nennt man View Selection Problem.

Aus der Definition lassen sich folgende Dinge schließen:

- Die Eingabegrößen beinhalten keine einzelne Datenbankinstanz. Trotzdem hängt die Lösung von den zum Datenbankschema gehörenden Statistiken und somit auch von Datenbankinstanzen ab. Die späteren Komplexitätsbetrachtungen beziehen sich allerdings auf die Größen des Datenbankschemas und des Workloads.
- Das View Selection Problem ist ein Optimierungsproblem. Das zugehörige Entscheidungsproblem, auf welches sich die späteren Komplexitätsbetrachtungen beziehen, ist das folgende: Gibt es eine Viewkonfiguration $\mathcal{V}$, so dass der Wert der Kostenabschätzungsfunktion $\mathcal{C}(\mathcal{R}, \mathcal{V}, \mathcal{Q})$ kleiner als eine gegebene Zahl k ist.
- Aufgrund der Speicherplatzbeschränkung ist es im allgemeinen keine Lösung alle Queries zu materialisieren.

Im vorangegangenen Abschnitt wurde das View Selection Problem formal definiert. Es wurden dafür die wichtigen Begriffe des Workload und der Viewkonfikuration eingeführt und es wurde auf Kostenabschätzungsfunktionen eingegangen.

3 Komplexität unter verschiedenen Annahmen

In [CHS01] und [CHS02] wird das View Selection Problem unter zwei verschiedenen Annahmen betrachtet. Es stellt sich heraus, dass beide Annahmen unterschiedliche Komplexitäten aufweisen, welche in diesem Abschnitt näher betrachtet und vergleichen werden. Alle Betrachtungen beziehen sich dabei auf so genannte SPJ-Queries. Dies sind Queries,

welche nur Selektionen, Projektionen und Joins, aber keine Aggregationen oder Gruppierungen beinhalten.

3.1 Complete Statistics Assumption

Bei der „Complete Statistics Assumption“ (CSA) wird angenommen, dass die zum Schema $\mathcal{R}$ gehörenden Statistiken exakt bekannt sind und somit auch die Größe jeder View exakt bestimmt werden kann. Man kann sich dies als ein Orakel vorstellen. Dieses Orakel kann in der Praxis durch viele Arbeitsschritte und die dadurch aufgestellten Statisiken approximiert werden.
Es ergeben sich unter der CSA folgende Komplexitäten (Beweise in [CHS02]):

- Die Größe einer View ist durch ein Exponential begrenzt, abhängig vom größtem Teilergebnis aller Queries im Workload.
- Die Anzahl der Views ist durch ein Exponential begrenzt, abhängig von der Größe von $\mathcal{R}$ und $\mathcal{Q}$.
- Das View Selection Problem ist in 4-fach exponentieller Zeit entscheidbar.

Die Ursache für die Begrenzung der Anzahl der Views durch ein Exponential ist die Projektion. Betrachtet man nur projektionsfreie Views, ist die Anzahl der Views durch ein quadratisches Polynom begrenzt, abhängig von der Größe von $\mathcal{R}$ und $\mathcal{Q}$. Die Beweisskizze dafür befindet sich in [CHS02].

3.2 Partial Statistics Assumption

Im Gegensatz zur CSA sind bei der „Partial Statistics Assumption“ (PSA) nicht alle Statistiken exakt bekannt. Somit ist diese Annahme praxisrelevanter. Allerdings hängt die Komplexität des VSP unter dieser Annahme sehr stark von der verwendeten Größenabschätzungsfunktion E ab. Verwendet man eine Funktion, die die Größe einer View „künstlich“ groß bestimmt, wird das Problem trivial, da es dadurch möglich sein kann, dass man aufgrund der Speicherplatzbeschränkung nur noch die Relationen selbst (falls nicht direkt darauf zugegriffen werden kann) oder auch gar nichts materialisieren kann. Verwendet man hingegen eine Funktion, die die Größe einer View „künstlich“ klein bestimmt, wird das Problem sehr komplex, da es so sehr viele verschiedene mögliche Viewkandidaten gibt.
Für die folgenden Komplexitätsabschätzungen werden daher Größenabschätzungsfunktionen aus der Praxis betrachtet, welche man als multiplikative Größenabschätzungsfunktionen bezeichnet.

Definition 3.1 (multiplikative Größenabschätzungsfunktion) *Eine Größenabschätzungsfunktion $E(\mathcal{R}, V)$, welche auf den Statistiken $\Sigma_{\mathcal{R}} = \{c_1, ..., c_p, N_1, ..., N_q\}$ basiert, wird als multiplikative Größenabschätzungsfunktion bezeichnet, wenn für jede View V gilt:*

$$E(\mathcal{R}, V) = c_1^{\gamma_1} \times ... \times c_p^{\gamma_p} \times N_1^{\delta_1} \times ... \times N_q^{\delta_q}$$

Dabei sind die c_i Selektivitätsfaktoren, die N_i Kardinalitäten der Releationen und die γ_i und δ_i natürliche Zahlen, welche von V abhängen.

Um stets eine optimale Lösung zu erhalten, müssen noch weitere Anforderungen an die Funktion E gestellt werden. Als Beispiel ist die „Anforderungen der großen Kardinalitäten“ zu nennen. Diese besagt, dass der Wert der Funktion E immer einen Mindestwert L annehmen muss. Ist also der von E bestimmte Wert kleiner als L, wird er auf den Mindestwert L gesetzt, ansonsten wird der ermittelte Wert beibehalten. Unter Berücksichtigung dieser Anforderungen an die Funktion E ergeben sich unter der PSA folgende Komplexitäten (Beweise in [CHS02] und [Chi02]):

- Die Größe einer View ist linear abhängig vom größten Teilergebnis aller Queries im Workload.
- Die Anzahl der Views ist durch ein Polynom begrenzt, abhängig von der Größe von $\mathcal{R}$ und $\mathcal{Q}$.
- Das View Selection Problem ist in exponentieller Zeit entscheidbar.

Somit ist das View Selection Problem in der Komplexitätsklasse NP.

3.3 Vergleich

Die Komplexitäten des View Selection Problem unter den beiden verschiedenen Annahmen sind in Tabelle 1 dargestellt:

	CSA	**PSA**
Größe einer View	durch ein Exponential begrenzt, abhängig vom größten Teilergebnis aller Queries im Workload	linear abhängig vom größten Teilergebnis aller Queries im Workload
Anzahl der Views	durch ein Exponential begrenzt, abhängig von der Größe von $\mathcal{R}$ und $\mathcal{Q}$	durch ein Polynom begrenzt, abhängig von der Größe von $\mathcal{R}$ und $\mathcal{Q}$
Entscheidbarkeit	in 4-fach exponentieller Zeit	in exponentieller Zeit

Tabelle 1: Vergleich der Komplexitäten des VSP unter der CSA bzw. der PSA

Man kann erkennen, dass durch die Existenz von genaueren Statistiken das Problem komplexer wird. Dies scheint auf den ersten Blick ein Widerspruch zu sein, allerdings gibt

es dafür eine einfache Erklärung: Bei der CSA liefert die Größenabschätzungsfunktion E stets die exakte Größe für jede View V. Bei der PSA wird nun ein Wert ermittelt, welcher durch die Annahme der großen Kardinalitäten größer ist als der eigentliche Wert. Durch zu groß abgeschätzte Views wird das Problem wiederum trivialer, wie im vorangegangenen Abschnitt bereits beschrieben wurde.

In diesem Abschnitt wurde das View Selection Problem unter der CSA und der PSA betrachtet. Man stellt fest, dass das Problem unter beiden Annahmen sehr komplex und somit der Suchraum der in Frage kommenden Views (Viewkandidaten) für eine optimale Lösung sehr groß ist.

4 Lösungsansatz

Im Folgenden wird ein Lösungsansatz für das View Selection Problem vorstellt, der den Suchraum der Viewkandidaten verkleinert. Dieser Ansatzt, welcher aus [TX04] entstammt, erzeugt aus gegebenen Queries entsprechende Viewkandidaten. Er wird dann in [XTZ06] verbessert, indem die Viewkandidaten nun auch mehr Relationen enthalten können als die Ausgangsqueries. Es werden dabei nur SPJ-Queries betrachtet.

Es wird dafür das Konzept des „Closest Common Derivator“ (CCD) eingeführt. Ein CCD stellt die maximale Gemeinsamkeit zwischen zwei Queries aus dem Workload dar und entspricht einem Viewkandidaten. Maximale Gemeinsamkeit bedeutet dabei möglichst viele Relationen und Operationen (Selektion, Projektion, Join) im CCD unterzubringen.
Um einen CCD zu erzeugen benötigt man zuerst die Querygraphen der beiden Ausgangsqueries. Nun wird basierend auf diesen Querygraphen der CCD (entspricht einem Viewkandidaten) erzeugt, indem nach und nach gewisse Regeln angewendet werden. Dabei handelt es sich zum einen um Regeln die gemeinsame Konten aufeinander abbilden, zum anderen existieren aber auch Regeln, welche die Knoten der einzelnen Queries aufteilen, so dass der CCD am Ende mehr Knoten haben kann als die Ausgangsqueries (was wiederum mehr Relationen bedeutet). Dies hat sich in der Praxis als nützlich erwiesen, wie die experimentellen Betrachtungen in [XTZ06] zeigen.
Mit den so entstandenen CCDs lassen sich danach die Ausgangsqueries nachweislich umschreiben.

Abbildung 1 zeigt den Gesamtprozess der CCD-Erzeugung:

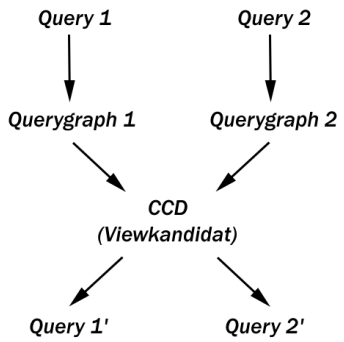

Abbildung 1: Schema zur Erzeugung eines CCD

In diesem Abschnitt wurde das Konzept des CCD als Lösungsansatz für das View Selection Problem vorgestellt. Dabei wird der Suchraum der Views für eine optimale Lösung verkleinert.

5 Zusammenfassung

Nach einer kurzen Einführung in die Problematik im ersten Abschnitt wurde das Problem im zweiten Abschnitt formal definiert. Bei der anschließenden Komplexitätsbetrachtung unter zwei verschiedenen Annahmen (CSA und PSA) wurde unter anderem festgestellt, dass das View Selection Problem unter der praxisrelevanten PSA in der Komplexitätsklasse NP liegt. Mit dem danach eingeführten Konzept des CCD existeriert bereits ein Lösungsansatz für das Problem, welcher den Suchraum der für die optimale Lösung relevanten Views verkleinert und somit vorhandene View-Selection-Algorithmen beschleunigt. Allerdings werden im Moment nur SPJ-Queries betrachtet, was eine erhebliche Einschränkung darstellt. In [XTZ06] wird aber bereits angekündigt, dass in Zukunft sowohl Aggregationen und Gruppierungen als auch Integritätsbedingungen des Schemas bei der Berechnung der CCDs berücksichtigt werden.

Literatur

[Chi02] Rada Chirkova. The view-selection problem has an exponential-time lower bound for conjunctive queries and views. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Seiten 159–168, New York, NY, USA, 2002. ACM Press.

[CHS01] Rada Chirkova, Alon Y. Halevy und Dan Suciu. A Formal Perspective on the View Selection Problem. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, Seiten 59–68, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[CHS02] Rada Chirkova, Alon Y. Halevy und Dan Suciu. A formal perspective on the view selection problem. *The VLDB Journal*, 11(3):216–237, 2002.

[TX04] Dimitri Theodoratos und Wugang Xu. Constructing search spaces for materialized view selection. In *DOLAP '04: Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, Seiten 112–121, New York, NY, USA, 2004. ACM Press.

[XTZ06] Wugang Xu, Dimitri Theodoratos und Calisto Zuzarte. Computing closest common subexpressions for view selection problems. In *DOLAP '06: Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, Seiten 75–82, New York, NY, USA, 2006. ACM Press.