4N-04

# 制約に基づく意味的キャッシング法の効率化

柳 奇亨† 石川 佳治‡ 北川 博之‡

†筑波大学 理工学研究科 ‡筑波大学 電子・情報工学系

## 1. はじめに

データベースにおけるキャッシング (caching) は，クライアント–サーバデータベース，データウェアハウス，分散および異種データベースのようなさまざまな分野で研究が行われてきた．キャッシングは，一般に問合せの間に時間的な局所性 (temporal locality) が存在するとき有効な問合せ処理の効率化の手法である．伝統的なキャッシングのアプローチでは，キャッシュされるデータ（通常，物理的なページあるいはタプル）は時間的な局所性の概念に基づいて管理されるが，近年，意味的な局所性 (semantic locality) の概念を用い，キャッシュデータを管理する意味的キャッシングのアプローチが提案されている[1,2].

意味的な局所性を用いるキャッシングのアプローチをここでは意味的キャッシング (semantic caching) と呼ぶが，その基本的なアイデアは，クライアントがサーバのデータ（本稿ではタプルの集合を想定する）をキャッシュする際に，各データに対し意味領域 (semantic region) を対応づけ，これにより，キャッシュされたデータの情報をコンパクトに表現しようというものである．新たに問合せが与えられた場合には，クライアントはキャッシュされた各データの意味領域と問合せの条件を比較することで，利用できるキャッシュ上のデータの情報を得ることができる．

既存の意味的キャッシングの研究として，[1]では意味情報を表現するために矩形領域 (rectlinear spatial region) を用いた手法が提案されている．また，[2]では，キャッシュされた各データに対し述語 (predicate) を対応づけることでキャッシュ情報の管理を行うことが提案されている．これらに対し，[3]では線形数値制約 (linear arithmetic constraint) を用いて意味情報を記述するアプローチを提案している．これは，キャッシュされたタプルを表現するために数値的な制約を用いるもので，制約データベース (constraint database) [4]の概念に基づいている．

本稿では，[3]のアプローチに基づく意味的キャッシング手法について，その処理の効率化について議論する．特に，意味領域のマッチング処理を効率化するための索引の利用について言及する．

## 2. 線形数値制約に基づく意味的キャッシング

先に述べたように，意味的キャッシングのアプローチでは，クライアントにおけるキャッシュの管理は意味領域に基づいて行われる．以下では，このようすを，例を用いて説明する．

サーバに以下のリレーションがあるとする．

Car (name, maker, year, displacement, price, tax)

ここでは，サーバがリレーショナル DBMS で，現時点のクライアントのキャッシュは空であるものと仮定する．ここで，ユーザから次のような問合せが与えられたとする．

$Q_1$:
SELECT name, year
FROM Car
WHERE displacement >= 1500 AND price <= 8000

クライアントキャッシュが空であるため，キャッシュマネジャは次のように $Q_1$ を単純な選択問合せに変換してサーバに送る（以下の問合せ例でも同様である）．

$Q_1$':
SELECT *
FROM Car
WHERE displacement >= 1500 AND price <= 8000

この問合せの結果，クライアントには条件を満たすタプルの集合がサーバから送られる．クライアントは得られたタプルの集合をもとに問合せ $Q_1$ の処理を行うが，その終了後，タプルの集合は破棄されず，クライアントのローカルキャッシュに格納されるものとする．このとき，キャッシュされるタプルの集合を記述する情報として，

$$\text{displacement} \geq 1500 \wedge \text{price} \leq 8000$$

という不等式の論理積（数値的な制約）を対応づける．この条件は，キャッシュされたタプル集合に対する意味領域として用いられる．

次に，以下のような問合せが与えられたとする．

$Q_2$:
SELECT name, price, tax
FROM Car
WHERE displacement >= 1200 AND price + tax <= 10000

この問合せの条件部分を見ると，問合せ $Q_1$ でキャッシュされた情報の一部が利用できることが見てとれる．これは，$Q_1$ によりキャッシュされたタプルの集合に対する意味領域が，$Q_2$ の意味領域とオーバラップしているためである．

$Q_1$ の問合せ条件 $q_1$ と $Q_2$ の問合せ条件 $q_2$ がオーバラップするかどうかは，$pq = q_1 \wedge q_2$ を計算し，その結果が空であるかどうかを調べることで決定できる． 実際には， displacement, price, tax を三つの次元とする空間上に各制約が示す領域を図示したとき，上の論理積はそれらの領域のオーバラップ部分に相当する．オーバラップが存在するかどうかは，$pq$ の論理式が満たす制約が空であるかどうかを調べることで判断できる[3]． 上の例の場合には，確かにオーバラップが存在し，

$pq$ = displacement $\geq$ 1500 $\wedge$ price $\leq$ 8000 $\wedge$ price + tax $\leq$ 10000

となる．この条件をローカルなキャッシュ上で評価することで，$Q_2$ の処理に必要でローカルキャッシュ上に存在するすべてのタプルを得ることができる．このような $pq$ を *probe query* と呼ぶ．

---

Efficient Semantic Caching Management Based on Linear Constraints
Gihyeong Ryu†, Yoshiharu Ishikawa‡, and Hiroyuki Kitagawa‡
†Master Program in Science and Engineering, University of Tsukuba
‡Institute for Information Sciences and Electronics, University of Tsukuba

上の問合せ $Q_2$ の処理において，必要なタプルをキャッシュ上から取り出すことは probe query で可能であるが，一方で，残りのタプルについては，サーバに要求する必要がある．このための問合せを *reminder query* と呼び，上の例の場合には $rq = \neg q_1 \wedge q_2$ で定義される．

意味的キャッシングでは，このように与えられた問合せに対して適切な probe query と reminder query を生成することで効率的な問合せ処理を実現する．この例ではキャッシュ上には $Q_1$ に関するキャッシュデータしか存在しなかったが，一般には複数の問合せ結果についてキャッシュデータが存在している．[3]では，上で述べた probe query と reminder query の生成法を一般的な場合について拡張している．

## 3. 制約データベースとしてのキャッシュ情報管理

本研究では，制約データベース (constraint database) [4]の概念を用いることでキャッシュデータの管理を行う．制約データベースとは，制約 (constraint) を直接的に表現・操作可能としたデータベースの総称である．ベースとなるデータモデルや用いられる制約の種類に応じてさまざまなカテゴリの制約データベースが存在する．本研究ではリレーショナルモデルをベースのモデルとし，制約については以下のような形式の制約を想定する．

$$a_1 x_1 + \cdots + a_p x_p \quad \theta \quad a_0$$

ここで，$x_i$ は変数，$a_i$ は有理数であり，$\theta \in \{=, \neq, <, \leq, \geq, >\}$ である．本研究で想定しているデータモデルでは，属性値としてこのような制約値を許すことで，キャッシュ情報のコンパクトな管理を実現する．

キャッシュ情報を効率的に管理するため，図 1 に示すように関連する次元（属性）をパーティション (partition) にまとめた形で表現する． 各行は一つの問合せの意味領域（タプル集合）に対応し，各パーティションはカラムに対応づけられる．このような次元の分割を行うことで，独立な制約条件を別個に扱えるため，キャッシュ利用の際の計算コストの削減に役立つことが知られている．

| id | 1st partition | 2nd partition | time |
|---|---|---|---|
| $q_1$ | displacement ≥1500 | price ≤ 8000 | $T_1$ |
| $q_2$ | displacement ≥1200 | price + tax ≤10000 | $T_2$ |

図 1: 制約を用いたキャッシュ情報の表現

前節で述べたように，与えられた問合せ $q$ が，問合せ $q_i$ についてキャッシュされているタプル集合の一部もしくはすべてを利用できるかどうかは，$q \wedge q_i$ という probe query の計算を行うことで判断できる．実際には，キャッシュには複数の問合せに対する複数個のタプルの集合がキャッシュされているため，probe query は

$$(q \wedge q_1) \vee \cdots \vee (q \wedge q_n)$$

で与えられる（$q_1, \ldots q_n$ はキャッシュされている各タプル集合に対応する問合せである）．ただし，実際にはパーティションへの分割を考慮することや，矛盾した制約の検出および冗長な制約の除去が必要である[3]．

## 4. 索引の使用による問合せ処理の効率化

前節に述べたように，制約を用いてキャッシュされたデータを表現するアプローチは，表現がコンパクトであり表現能力も高いという性質を有している反面，キャッシュが利用可能かどうかをチェックする場合などの計算コストが無視できないという問題点が存在する．処理コストの基本的な要因は，キャッシュされた各タプル集合 $q_i$ に対し，与えられた問合せ $q$ との論理積 $q \wedge q_i$ を求めるコストが高く，さらにこれが，キャッシュ中に存在する $n$ 個のタプル集合のすべてについて行われなければならないことによる．そのようなコストを削減する一つの策として，3 節で述べたような次元の分割の手法がある．分割を用いることにより，計算コストのうち，特に制約条件にの次元数に起因する部分の削減が可能となる．

計算コストのうち，キャッシュ中のタプル集合の総数 $n$ に起因する部分については，適切な索引手法を活用することでその削減を図ることができる．索引手法の活用により，与えられた問合せ条件 $q$ とオーバラップする（すなわち $q \wedge q_i$ が空でない）可能性があるような $q_i$ の候補を効率よく列挙することができれば，大幅に問合せ処理コストを削減できる可能性がある．

制約データベースに対する索引手法は，[5]において述べられているが，基本的には時区間データベース，空間データベースにおいて開発された索引手法を制約データベースにおける問合せ処理に適用することになる．ただし，制約データベースでは制約の領域が閉じていない場合（半平面）があることなどで，時区間・空間データベースのための索引手法が直接的に利用できない場合もある．[5]によれば，制約の次元数が 1 ないし 2 次元の場合には，すでに効率的な索引手法が提案されていることから，本研究では制約の分割を行うことで，1 ないし 2 次元のパーティションの索引づけを可能とする．3 次元以上の次元数のパーティションについては，より低次元のパーティションに分解することが考えられる．この場合には，索引の使用だけではオーバラップかどうかは正確には判断できず，オーバラップする可能性を知ることのみができるため，オーバラップ候補の検出のためのフィルタリング処理として用いることになる．

## 5. まとめと今後の課題

本稿では，線形制約によりキャッシュ情報を記述する意味的キャッシング手法について，そのアプローチについて述べ，効率化のための分割の適用，索引の活用について考察した．今後はこれらの内容についてさらに検討を行い，アルゴリズムの開発やシミュレーションによる評価を行いたい．

## 参考文献


[1] S. Dar, M.J. Franklin, B.T. Jósson, D. Srivastava, and M.Tan: "Semantic Data Caching and Replacement", in *Proc. of VLDB*, pp. 330–341, Mumbai, India, Sept. 1996.

[2] M. Keller and J. Basu: "A Predicate-based Caching Scheme for Client-server Database Architectures", *VLDB Journal*, 5(1): 35–47, Jan. 1996.

[3] Y. Ishikawa and H. Kitagawa: "A Semantic Caching Method Based on Linear Constraints", in *Proc. of 1999 Intl. Symp. on Database Applications in Non-Traditional Environments (DANTE'99)*, pp. 174–181, Kyoto, Japan, Nov. 1999.

[4] V. Gaede and M. Wallace: "An Informal Introduction to Constraint Database Systems", in V. Gaede et al. (eds.), *Constraint Databases and Applications*, Vol. 1191 of LNCS, pp. 7–52, Springer-Verlag, 1997.

[5] E. Bertino et al.: *Indexing Techniques for Advanced Database Systems*, Kluwer Academic Publishers, 1997.