サービス指向ルータ向け問合せ処理用ハードウェアの検討

松谷 宏紀[1]

概要：本論文では、サービス指向ルータにおける問合せ処理高速化のためにハードウェアによる SQL キャッシュを検討する。まず、プロセッサ単体の問合せ処理性能を見積もり、次に、ハードウェアによる SQL キャッシュのための設計方針を決め、問合せの多様性とカテゴリ分けを行う。これらの議論を踏まえ、ハードウェアによる SQL キャッシュとして、集合関数の結果をキャッシュする結果キャッシュ、及び、直近 1 時間の全レコードを保持するバッファキャッシュを設計し、待ち行列理論を用いて性能向上率を見積もる。最後に、ハードウェア量の考察を行い、また、C 言語で実装した SQL キャッシュシミュレータを用いて提案機構の有用性を確認する。

A Case for Query Processing Hardware for Service-Oriented Routers

Hiroki Matsutani[1]

Abstract: This paper introduces a hardware-based SQL cache mechanism for accelerating the query processing on service-oriented routers. First, query processing performance using a single processor is estimated. Then, design policies for the hardware-based SQL cache are mentioned and SQL queries for the service-oriented routers are discussed in detail. Based on the discussion, the hardware-based SQL cache is designed with two parts: the result cache that stores the recent query results and the buffer cache that stores the recently inserted records. A performance model based on the queueing theory is built. Finally, feasibility of the proposed SQL cache is discussed and the performance improvement is also verified by using a C-level SQL cache simulator.

1. はじめに

インターネットの普及と仮想化及びクラウド技術の発展によって、様々な情報サービスが次々と登場し続けており、今後も豊かな情報サービスに対する需要が高まってくると考えられる。このような需要に対し、インターネットルータにおいてネットワークコンテンツを蓄積・解釈可能なサービス指向ルータ [1] が注目を浴びている。サービス指向ルータでは、パケットの意味情報の解析結果により情報の授受・検索・共有を実現でき、新たなサービスを創出する可能性を秘めている。現在は、さらなるサービス親和性を高めるための分散協調動作、及び、インフラとしての側面も追求したサービス指向ルータの研究が行われている。具体的には、1) サービス指向ルータの情報抽出の基本機能や方式、2) メモリ効率の良いスケーラブルなソフトウェアシミュレータの活用、展開、統合することで実働するサービス指向ノードを構築し、ネットワークの仮想化のプラットフォームの開発が行われている [1]。

サービス指向ルータにおける重要な機能の 1 つにデータベースへの問合せ処理がある。これはサービス指向ルータに蓄積されたネットワークコンテンツをクライアント側から検索可能とするために必須の機能であり、高い問合せ処理性能が要求される。そこで、本論文では、サービス指向ルータにおける問合せ処理高速化のためにハードウェアによる SQL キャッシュを検討する。まず、プロセッサ単体の問合せ処理性能を見積もり、次に、ハードウェアによる SQL キャッシュのための設計方針を決め、問合せの多様性とカテゴリ分けを行う。これらの議論を踏まえ、ハードウェアによる SQL キャッシュとして、集合関数の結果をキャッシュする結果キャッシュ、及び、直近 1 時間の全レコードを保持するバッファキャッシュを設計し、待ち行列理論を用いて性能向上率を見積もる。最後に、ハードウェア量の考察を行い、また、C 言語で実装した SQL キャッシュシミュレータを用いて提案機構の有用性を確認する。

本論文の構成は以下の通りである。まず、2 章にて、プロセッサ単体での SQL 問合せ処理性能を測定する。3 章にて、ハードウェア SQL キャッシュの設計について述べ、キャッシュヒット率と問合せ処理性能をモデル化する。4 章にて性能及びハードウェア量に関する予備評価を示し、5 章にて本論文をまとめる。

[1] 慶應義塾大学 理工学部
Faculty of Science and Technology, Keio University, Japan

表 1 実験に使用したデータベースの定義

フィールド名	型	説明
ts	TIMESTAMP	INSERT 時のタイムスタンプ
saddr	INTEGER	送信元 IPv4 アドレス
daddr	INTEGER	宛先 IPv4 アドレス
proto	INTEGER	プロトコル番号
sport	INTEGER	送信元ポート番号
dport	INTEGER	宛先ポート番号
data	TEXT	ペイロードデータ

2. プロセッサによる SQL 問合せ処理性能

本研究の目的は、サービス指向ルータにおける問合せ処理の高速化である。そのために、まずは、プロセッサ単体（ハードウェアによる高速化無し）で RDBMS を動作させたときの性能を測定する。

2.1 データベースの定義

サービス指向ルータにおいて、IP パケットからどのようなデータを抽出し、データベースに登録するかを決める必要がある。これまでにサービス指向ルータを用いた様々なアプリケーションが提案および研究されてきており [1]、今後も新たなサービスが開拓され続けると考えられる。したがって、サービス指向ルータを用いて提供するサービス内容を鑑み、注意深くデータベースを定義する必要がある。

一方、本研究の目的はデータベースの問合せ処理の高速化である。よって、アプリケーションを考慮したデータベースの定義自体は検討の対象外とし、**表 1** に示す最低限のフィールドのみをデータベースに登録するものとして議論を進める。なお、proto フィールドは IP ヘッダに含まれるプロトコル番号であり、代表的なものとしては ICMP が 1、TCP が 6、UDP が 17 である。data はアプリケーション層のペイロードデータであり、例えば、HTTP（dport=80）の GET/POST リクエスト、SMTP（dport=25）の SMTP コマンド、POP3（dport=110）の POP3 コマンドの一部を格納するものとする。

2.2 評価環境

2.2.1 対象プロセッサ

ルータに搭載するプロセッサとしては、組込み用プロセッサ

表 2 CPU のシミュレーションパラメータ

	ハイエンド CPU	組込み CPU
アーキテクチャ	x86-64	ARMv7
動作周波数	2.66GHz	0.80GHz
L1I キャッシュ	32kB	32kB
L1D キャッシュ	32kB	32kB
L2 キャッシュ	256kB /core	512kB /core
ゲスト OS	Linux 2.6.22.9	Linux 2.6.38.8
コンパイラ	GCC 4.4.4 (-O2)	GCC 4.6.2 (-O2)
RDBMS	SQLite 3.7.15.2 (in-memory)	SQLite 3.7.15.2 (in-memory)

表 3 実験に使用した SQL 文

番号	SQL 文
OP.1	INSERT INTO sor VALUES (ランダムデータ)
OP.2	SELECT COUNT(*) FROM sor
OP.3	SELECT COUNT(*) FROM sor WHERE proto=6
OP.4	SELECT ts FROM sor
OP.5	SELECT ts FROM sor WHERE proto=6
OP.6	SELECT ts,saddr,daddr,proto,sport,dport FROM sor
OP.7	SELECT ts,saddr,daddr,proto,sport,dport FROM sor WHERE proto=6
OP.8	SELECT * FROM sor
OP.9	SELECT * FROM sor WHERE proto=6

からハイエンドプロセッサの利用まで考えられる。使用するプロセッサによって、SQL 問合せ処理のスループットは大きく変化するため、以下の 2 種類のプロセッサを想定して予備評価を行う。

- ハイエンドプロセッサ（アーキテクチャ：x86-64、動作周波数：2.66GHz）
- 組込みプロセッサ（アーキテクチャ：ARMv7、動作周波数：0.80GHz）

2.2.2 フルシステムシミュレータ

様々なアーキテクチャ上において効率的に予備評価を行うため、フルシステムシミュレータ gem5 [2] を用いた。プロセッサ、メモリ階層（L1、L2、主記憶）、結合網の正確なシミュレーションに加え、シミュレータ上でゲスト OS（Linux 等）及び開発環境（GCC 等）が動作するため、実機に近い環境をシミュレーションできる。**表 2** にハイエンド環境及び組込み環境のシミュレーションパラメータを示す。

2.2.3 関係データベース管理システム

関係データベース管理システム（Relational Database Management System、RDBMS）[3] は、関係モデルに基づいたデータベース管理システムである。商用の RDBMS に加え、オープンソースの RDBMS として MySQL、PostgreSQL、SQLite [4] などが広く知られており、多くの場合、問合せ言語として SQL を用いる。

一般的に、RDBMS における問合せ処理のボトルネックはディスク I/O である[*1]。RDBMS においてディスク I/O を減らすために、読み込みブロックをメモリにキャッシュする方式（バッファキャッシュ）や問合せ結果をメモリにキャッシュする方式（結果キャッシュ）が利用されている。また、汎用のメモリキャッシュシステム（例えば、Memcached [6]）との併用も可能である。

一方、本研究では RDBMS をインターネットルータ上の制御用プロセッサで動作させる点を考慮し、組込み用途で利用可能な SQLite [4] を主記憶上（in-memory）で動作させる。保持できるデータ量は主記憶のサイズによって制限されるものの、ディスク I/O による遅延を考慮する必要は無いというメリットがある。

2.2.4 登録及び問合せ処理の内容

プロセッサ単体で**表 3** に示す登録処理及び問合せ処理を実行し、処理に要した時間を測定する。

OP.1 は登録処理である。表 1 で定義したフィールドに対し、ランダムにデータを生成し、sor（Service-Oriented Router の略）という名前のデータベースに登録する。data フィールドに関しては、HTTP の GET リクエストに含まれるファイルのパス名を想定して 128-Byte のランダムな文字列を登録した[*2]。

OP.2〜OP.9 は sor に対する問合せ処理である。そのうち OP.2 と OP.3 は集合関数 COUNT(*) を用いた例である。OP.2 はデータベース中の全レコード数、OP.3 は WHERE 句を用いた条件付き参照の例として TCP（proto=6）に対応するレコード数を返す。

一方、OP.4〜OP.9 は単純な表の取り出し処理であり、OP.4〜OP.5 は単一フィールド（ts）の取り出し、OP.6〜OP.7 はパケットのヘッダ情報（ts、saddr、daddr、proto、sport、dport）の取り出し、OP.8〜OP.9 はレコード全体の取り出しである。

[*1] 主記憶（DRAM）のアクセス時間は 50n〜70n 秒程度であるのに対し、磁気ディスクのアクセス時間は 5m〜20m 秒である [5]。

[*2] GET リクエストに含まれるファイルのパス名はほとんどの場合 128-Byte 以内に収まると考えた。

表 4 問合せの CPU 処理時間（n=36,000、単位は [msec]）

	ハイエンド CPU	組込み CPU
OP.2	1.442	2.962
OP.3	43.669	127.141
OP.4	58.224	179.947
OP.5	46.704	137.069
OP.6	152.494	513.986
OP.7	59.201	178.302
OP.8	181.615	615.747
OP.9	64.033	192.199
平均 $\bar{x}$	75.923	243.419
標準偏差 σ	56.136	195.377

2.2.5 測定用プログラム

SQLite の C 言語用 API [4] を用いて OP.1〜OP.9 を実装し、表 2 に示したシミュレーション環境でコンパイル及び実行した。登録するレコード数として 10、100、1,000、10,000、36,000、100,000 個の場合それぞれについて処理時間を測定することで、レコード数の増加に対する登録及び問合せ処理の時間の変化を確認した。

データベースへの登録処理（OP.1）については、ランダムに生成した列データを含む INSERT 文を引数に、登録するレコードの個数だけ sqlite3_exec() 関数を実行した。

問合せ処理（OP.2〜OP.9）については、まず、sqlite3_prepare() 関数を用いて問合せ内容を内部コードに変換し、sqlite3_step() 関数によって問合せを行い、sqlite3_column_int() 関数もしくは sqlite3_column_text() 関数を用いて各フィールドの値を取り出す。

以上の OP.1〜OP.9 について、UNIX の gettimeofday() 関数を用いて実行時間を測定した。

2.3 登録及び問合せ処理時間

図 1 にプロセッサ単体の問合せ処理時間を示す。X 軸は登録レコード数、Y 軸は OP.1〜OP.9 の処理時間である。フルシステムシミュレータを用いたハイエンド環境（**図 1(a)**）及び組込み環境（**図 1(b)**）に加え、参考のため、実際の PC 上での測定結果（**図 1(c)**）も示す。シミュレーションによるハイエンド環境の結果と実機による結果を比べると、誤差はあるものの実機に近い値が出ていることが分かる。図 1 の結果より、レコード数にほぼ比例して、登録及び問合せの処理時間が増加していることが分かる。また、ハイエンドプロセッサ環境と組込みプロセッサ環境を比べると、その動作周波数の違いから数倍の性能差があるものの、レコード数と処理時間の傾向は類似していることが分かる。

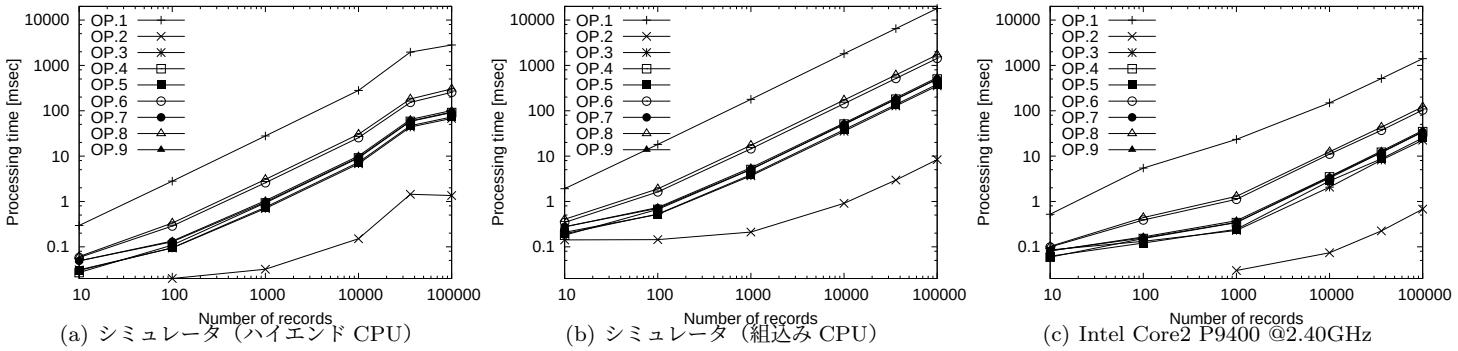
以降の議論において、データベース中に存在するレコード数をどの程度として見積もるかが重要になってくる。**表 4** にレコード数 n=36,000 の場合の問合せ処理時間、その平均及び標準偏差を示す（ここで n=36,000 とした理由については 3.4 節にて議論する）。

2.4 問合せ処理スループット

表 4 で求めた処理時間のパラメータを用いて、プロセッサ単体の場合に単位時間当たりどれだけ問合せを処理できるか（処理スループット）を見積もる。

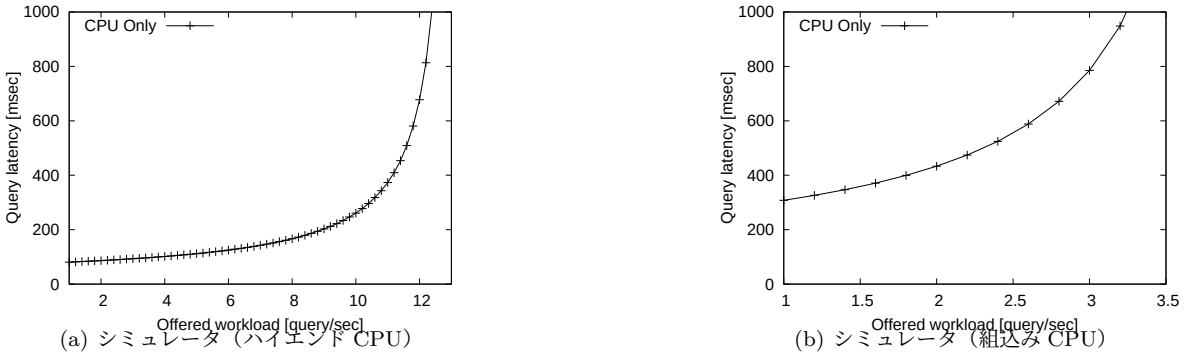
ここでは簡易的に待ち行列モデルを用いて問合せの処理スループットを計算する。待ち行列モデルには M/G/1 を用いた。これは、問合せ要求の到着率 λ はポアソン過程にしたがい、サービス時間は表 4 に示した平均と標準偏差にしたがう一般分布、窓口（問合せを処理するプロセッサ）は 1 つのモデルである。まず、表 4 に示した平均 $\bar{x}$ と標準偏差 σ よりサービス変動係数 $c = \sigma/\bar{x}$ を求めた。次に、問合せ要求の到着率 λ を変化させながら、Little の公式 [7] を用いて、問合せが要求されてから問合せが完了するまでの平均時間（平均待ち時間 + 平均問合せ処理時間）を計算した。

図 2 にこのようにして計算したプロセッサ単体の問合せ処理スループットを示す。レコード数は n=36,000 の場合を想定し



(a) シミュレータ（ハイエンド CPU）　(b) シミュレータ（組込み CPU）　(c) Intel Core2 P9400 @2.40GHz

図 1　登録及び問合せの CPU 処理時間（横軸はレコード数、縦軸は処理時間 [msec]）。(b) 及び (c) はフルシステムシミュレータによる実行結果をそのまま載せただけであり、実際の製品を用いたベンチマーク結果ではないことに注意されたい。



(a) シミュレータ（ハイエンド CPU）　(b) シミュレータ（組込み CPU）

図 2　問合せ処理のスループット（CPU のみ、n=36,000、M/G/1 モデルを使用）

た。簡易的なモデルを用いた見積りではあるが、実際の処理スループットは 3〜11 [query/sec] 程度であることが予想されるため、次章で述べるようなハードウェアによる問合せ処理の高速化が必要と言える。

3. ハードウェア SQL キャッシュ

ハードウェア SQL キャッシュを設計するに際し、まずは以下の 3 点を仮定する。

- サービス指向ルータ（以降、単にルータ）の制御用プロセッサ上で RDBMS が in-memory で動作する。
- データベースを利用するクライアントマシンが、ルータに対し問合せ処理を要求する。
- ハードウェア SQL キャッシュは、プロセッサの前段に挿入され、キャッシュがヒットした場合はプロセッサを介さずに問合せ結果をクライアントに返す。キャッシュミスの場合は、通常通りプロセッサが問合せを処理する。

3.1 設計方針

上述の通り、IP パケットの情報を蓄積するルータ側に対し、データマイニング等を行うクライアントマシンが問合せ処理を要求する。この場合、ルータに組み込まれる制御プロセッサは、クライアントマシンに比べて処理性能及びストレージ容量の点で不利であると考えられる。また、複数のクライアントマシンからルータ側へ問合せ要求が集中する可能性もある。

そこで、本論文では以下の方針にしたがいハードウェア SQL キャッシュを設計することにする。

(1) クライアント側でできる処理はクライアント側で行う。
(2) 分割可能な問合せ処理は、クライアント側が問合せを複数回行うことで実現する。
(3) ストレージ容量の制限から、ルータ側で多量のデータを永続的に保持するのではなく、永続的なデータ保持にはクライアント側のストレージを使用する。

設計方針 (1) について、クライアント側でできる処理の代表例は ORDER BY 句を用いた問合せ結果のソートである。このような処理を処理能力の高いクライアント側で実行することで、負荷が集中しやすいルータ側の負荷を軽減できる。

設計方針 (2) については、例えば、WHERE 句として'A' OR 'B' を含む問合せは、クライアント側が WHERE 句として'A' を指定した問合せと'B' を指定した問合せに分けて実行し、クライアント側で 2 つの問合せ結果の和集合をとることで実現する。また、入れ子問合せ処理も複数の問合せに分割可能である。設計方針 (2) によって問合せ処理を簡素化できるうえ、設計方針 (1) にも合致するためルータ側の負荷を軽減することができる。

設計方針 (3) にしたがい、クライアント側は 1 日もしくは 1 時間毎など比較的大きな時間粒度でルータに問合せを行い、その結果をクライアント側の大規模なストレージに保持する。そのため、ルータ側への問合せにおける探索条件（ts フィールドによる範囲探索）には時間的局所性（直近 1 日もしくは直近 1 時間の範囲探索が多発するなど）が生じやすい。

3.2 問合せの多様性とカテゴリ分け

3.2.1 Class-0 問合せ

SQL を用いた問合せの基本構造は、SELECT 句を用いた選択リスト、FROM 句を用いた表参照リスト、WHERE 句を用いた探索条件から構成される。SQL-92 で標準化された機能は多岐に渡り、極めて柔軟な問合せ処理が可能となっている。ここでは、SQL-92 でサポートされている問合せ機能を Class-0 と定義する（**表 5**）。

3.2.2 Class-1 問合せ

一方、本研究ではこの問合せ処理を付加的ハードウェアを用いて高速化することを目的とする。Class-0 ではユーザによる利便性を考慮した柔軟性の高いインタフェースを提供するが、それを解釈するための処理が必要となる。結局のところ、問合せ処理の柔軟性と性能の間にはトレードオフの関係があるため、ここでは Class-0 に必要最低限の制限を加えることで高速化していく。

Class-1 問合せは、Class-0 問合せに以下に示す制限を加えたものである（表 5）。

- SELECT 句として、指定可能な列（ts、saddr、daddr、proto、sport、dport、data）の任意の組合せ、もしくは、代表的な集合関数である COUNT(*) に対応する。
- WHERE 句として、3.1 節で述べた設計方針 (2) にしたがい、論理演算子は AND のみに対応し、OR には対応しない。ts フィールドは BETWEEN 句を用いた範囲探索、それ以外のフィールドに関しては等号（=）を用いた等号探索に対応する。
- 設計方針 (1) 及び (2) にしたがい、ORDER BY 句によるソート処理や入れ子問合せには対応しない。

表 5 問合せの多様性とカテゴリ分け Class 0、1、2、3（Class-1：本機構で応答可能な問合せ、Class-2：キャッシング対象の問合せ、Class-3：高頻度な問合せの例）

Class-0	SELECT 句	指定可能な列（ts,saddr,daddr,proto,sport,dport,data）の任意の組合せ、任意の集合関数
	WHERE 句	論理演算子（AND、OR）、比較演算子、BETWEEN 句、IN 句、LIKE 句を用いたすべての組合せ
Class-1	SELECT 句	指定可能な列（ts,saddr,daddr,proto,sport,dport,data）の任意の組合せ もしくは、集合関数 COUNT(*)
	WHERE 句	以下の 6 条件を論理演算子 “AND” で結合した条件式のみ ts BETWEEN INT_{32} AND INT_{32} ※ INT_{32} として time_t 構造体を想定 saddr = INT_{32} ※ INT_{32} として IPv4 アドレスを想定 daddr = INT_{32} proto = INT_8 ※ INT_8 としてプロトコル番号を想定 sport = INT_{16} ※ INT_{16} として TCP/UDP ポート番号を想定 dport = INT_{16}
Class-2	SELECT 句	COUNT(*) の場合
	WHERE 句	以下の条件を含む Class-1 WHERE 句 ts BETWEEN INT_{32} AND INT_{32} ※直近 1 日で 10 分単位の指定に限定、144 通り
	SELECT 句	指定可能な列（ts,saddr,daddr,proto,sport,dport,data）の任意の組合せ
	WHERE 句	以下の条件を含む Class-1 WHERE 句 ts BETWEEN INT_{32} AND INT_{32} ※直近 1 時間で 10 分単位の指定に限定、6 通り
Class-3	SELECT 句	以下の 4 種類のいずれか COUNT(*) ts,saddr,daddr,proto,sport,dport data *
	WHERE 句	以下の 4 条件を論理演算子 “AND” で結合した条件式のみ ts BETWEEN INT_{32} AND INT_{32} ※直近 1 時間で 10 分単位の指定に限定、6 通り daddr = INT_{32} ※少数のホストに限定、16 通り proto = INT_8 ※ TCP に限定、1 通り dport = INT_{16} ※一部の well-known サービスに限定、4 通り

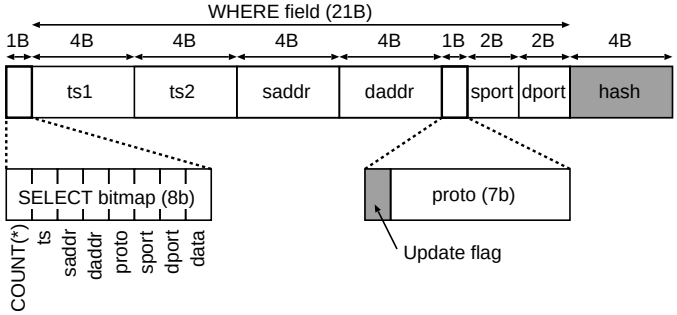


図 3 Class-1 問合せのためのデータ構造（SELECT 句の指定に 1-Byte、WHERE 句の指定に 21-Byte）

本論文で提案するハードウェアによる SQL キャッシュはすべての Class-1 問合せに応答できるように設計する。Class-1 の簡略化によって Class-1 問合せは**図 3** に示す 26-Byte のデータとして表現できる。具体的には、

- 1-Byte 目は SELECT 句による選択リストに対応し、8-bit ビットマップの対応するビットを 1 にすることで取り出したい列を指定する。なお、COUNT(*) を指定した場合は他の列は指定できない。
- 次の 21-Byte は WHERE 句による探索条件に対応し、BETWEEN 句を用いた ts の範囲探索（4-Byte フィールドを 2 個使用）、saddr、daddr、proto、sport、dport の等号探索の比較値として使用される。なお、比較値として全ビットを 1 とした場合はその列の探索条件は*として扱われる。

なお、proto フィールドの最上位 1-bit は Update フラグとして使用する。Update フラグについては 3.3 節で述べる。最後の 4-Byte にある hash 値についても 3.3 節で述べる。

3.2.3 Class-2 問合せ

本ハードウェア機構はすべての Class-1 問合せに対応しているが、Class-1 の問合せ処理の組合せは膨大な数に及ぶため、Class-1 の問合せ結果をすべてキャッシュしたとしてもキャッシュ容量の制限からキャッシュヒットはほとんど見込めない。そこで、問合せを行うクライアント側でキャッシュがヒットしやすいように問合せを工夫する（場合によっては複数の問合せを実行する）必要がある。

Class-2 問合せでは、Class-1 に加え、ts フィールドの指定方法に制限を加えている。ts フィールドを用いた範囲探索においてはミリ秒単位などの細かい粒度の範囲指定よりも、10 分単位などの比較的大きな粒度の範囲指定のほうが利用頻度が高いと判断したため、Class-2 では WHERE 句の ts1 フィールドを 10 分単位の指定とし、ts2 フィールドは ts1 に 10 分を加えた値に制限している。例えば、30 分間の範囲探索は 10 分単位の範囲探索を 3 回実行することで実現できる（3.1 節の設計方針 (2) に対応）。

本ハードウェア機構はすべての Class-1 問合せに対応しているが、クライアント側が問合せの際に ts フィールドの指定方法を工夫し可能な限り Class-2 化することでキャッシュヒットの可能性及び性能が大幅に向上することを 4.2 節の評価で示す。

なお、Class-2 問合せを高速化するキャッシュ機構の詳細については 3.3 節及び 3.4 節で説明する。

3.2.4 Class-3 問合せ

ここまで、WHERE 句の比較値として任意の値が指定されるものとして議論を進めてきたが、実際の IP ヘッダにおいて利用されるフィールドの値には強い局所性があると考えられる。例えば、

- proto の値として、ICMP（proto=1）、TCP（proto=6）、UDP（proto=17）の利用頻度が高い。
- sport よりも dport をキーにした検索のほうが利用されやすい。また、代表的なサービスは HTTP（dport=80）、SMTP（dport=22）、POP3（dport=110）、SSH（dport=22）など種類はそれほど多くない。
- daddr はそのルータが所属する特定ネットワーク内の公開サーバに限定されることが多い。

4.2 節で示すハードウェアによる SQL キャッシュのシミュレーション評価では、高頻度な問合せ例として Class-3 問合せが 25%、50%、75%含まれる場合について評価する。

3.3 結果キャッシュ

結果キャッシュでは、表 5 の Class-2 に合致する集合関数 COUNT(*) を用いた問合せ結果（OP.2〜OP.3 に対応）をキャッシュする。

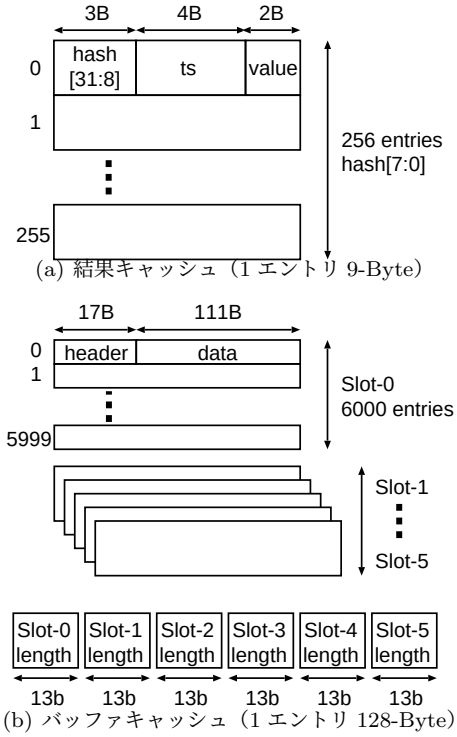
3.3.1 キャッシュポリシ

具体的には、Class-1 の制約に加え、BETWEEN 句を用いた ts の指定が 10 分単位、かつ、直近 1 日以内の場合のみ問合せ結果をキャッシュする。例えば、BETWEEN ’YYYY 年 MM 月 DD 日 HH 時 20 分’ AND ’YYYY 年 MM 月 DD 日 HH 時 30 分’、かつ、これが直近 1 日以内の場合キャッシュする。

3.3.2 キャッシュ構造

図 4(a) に結果キャッシュのデータ構造を示す。各キャッシュエントリは、クライアントが生成した問合せのハッシュ値[*3]（32-bit のうち上位 24-bit）、キャッシュ時のタイムスタンプ（32-bit）、結果の値（16-bit）から構成され、図 4(a) の例では 256 エントリ分のキャッシュを保持している。問合せに含まれるハッシュ値下位 8-bit をインデックスとし、対応するエントリのハッシュ値上位 24-bit と比較し、一致した場合は、キャッシュヒットとして、プロセッサを介さずに結果の値をクライア

[*3] 図 3 の hash フィールドに対応する。



(a) 結果キャッシュ（1 エントリ 9-Byte）

(b) バッファキャッシュ（1 エントリ 128-Byte）

図 4 Class-2 問合せのためのキャッシュ構造

ント側へ返す。

3.3.3 ハッシュ値の生成

問合せのハッシュ値は、クライアント側が問合せを生成する際、図 3 の上位 22-Byte の値を用いて 32-bit のハッシュ値を計算し、問合せの最下位 4-Byte に格納することでルータ側に引き渡される。したがって、ルータ側ではハッシュ値の計算は行わない。

ハッシュ値の計算には MD5 や SHA1 などの暗号をもとにしたハッシュ関数が利用できるが、ここでは MurmurHash [8] を使用した。MurmurHash は低い計算コストで良好な一様性を持ったハッシュ関数であり、可変長の入力データから 32-bit（もしくは 128-bit）のフットプリントを生成する。

3.3.4 キャッシュデータの陳腐化について

キャッシュされたデータの陳腐化についてはルータ側では一切関知しない。その代わり、キャッシュがヒットした場合、クライアントには、問合せ結果に加え、そのキャッシュ値が生成されたタイムスタンプ（図 4(a) の ts フィールド）を返すようにした。結果を受け取ったクライアントがデータが古いと判断した場合、クライアントは Update フラグ（図 3 の Update ビットに対応）を付けて、問合せを再実行する。Update フラグが付いた問合せでは、キャッシュを見には行かず、プロセッサによって直接問合せ処理を行う。この問合せ結果はキャッシュされるため、これによって結果キャッシュの中身が更新される。

ルータ側でキャッシュの陳腐化について対処しようとすると、ルータ側で、現在どのデータが複製（キャッシュ）されているかを管理し、INSERT によるデータベースの更新によってキャッシュされたデータが陳腐化した場合は逐一キャッシュ値を無効化、もしくは、更新する必要が生じる。一般的にこのような処理は極めて複雑であり、かつ、後々利用されないキャッシュについても値が最新になるように常にメンテナンスし続ける必要があるため無駄が多いと判断した。それよりは、計算能力の高いクライアント側に陳腐化の判断を任せ、クライアント主導でルータ側キャッシュの更新を行うほうがルータ側の負担を軽減できると判断した（3.1 節の設計方針 (1) に対応）。

3.4 バッファキャッシュ

バッファキャッシュでは、表 5 の Class-2 に合致する表の取り出し処理（OP.4〜OP.9 に対応）をキャッシュする。

3.4.1 キャッシュポリシ

Class-1 の制約に加え、BETWEEN 句を用いた ts の指定が 10 分単位、かつ、直近 1 時間以内の場合、範囲内の全レコードをキャッシュする。つまり、SELECT 句でどのような列が指定されていたとしても、時間の探索条件が Class-2 を満たしている場合、時間の探索範囲（10 分間）に含まれる全レコードを一度バッファキャッシュに格納する。

3.1 節の設計方針 (3) で議論したとおり、ルータ側のストレージ容量はクライアント側に比べて制限されるため、ルータ側で多量のデータを永続的に保持するのではなく、クライアント側は 1 時間毎など比較的大きな時間粒度でルータに問合せを行い、その結果をクライアント側の大規模ストレージに保持するほうが現実的である。したがって、ルータ側において直近 1 時間のレコードのみキャッシュするだけでもキャッシュヒット率はかなり向上するものと考えられる。

表 6 問合せキャッシュの DMA 転送時間（n=36,000、キャッシュは 200MHz 動作、単位は [usec]）

	転送データサイズ（期待値）	転送時間（期待値）
OP.2	4	0.005
OP.3	4	0.005
OP.4	144,000	180.000
OP.5	14,400	18.000
OP.6	612,000	765.000
OP.7	61,200	76.500
OP.8	4,608,000	5,760.000
OP.9	460,800	576.000

3.4.2 キャッシュ構造

図 4(b) にバッファキャッシュのデータ構造を示す。各レコードは、表 1 に示した全フィールドを保持するものとする。これには ts、saddr、daddr、proto、sport、dport を含む 17-Byte 分のヘッダ情報及びペイロード data の 111-Byte 分が含まれ、レコード当たり合計 128-Byte となる。

図 1 の測定結果より、INSERT は比較的時間のかかる処理であるため、ここでは 1 秒間に最大 10 レコードが追加されるものとする。時間の探索範囲（Slot と呼ぶ）は 10 分単位であるため、Slot 当たり最大 6,000 レコード保持する必要がある。直近 1 時間分の全レコードをキャッシュするには 6 個分の Slot を確保する必要があるため、メモリ容量は合計 4,500kByte となる。

3.4.3 問合せ結果のフィルタリング

上述のとおり、バッファキャッシュでは SELECT 句の列指定に依らず、Class-2 条件を満たしている場合、単位時間（10 分間）分の全レコードをキャッシュする。ただし、クライアント側に問合せ結果を返す際は、SELECT 句による列指定にしたがい必要なフィールドのみ返す必要がある。これはバッファキャッシュへのストライドアクセスによって実現する。具体的には、バッファキャッシュの中身を DMA 転送する際、指定された列の値のみ取り出すように図 4(b) においてバッファキャッシュを一定間隔でストライドアクセスすることによって、指定されていない列を転送しないようにする。また、WHERE 句における等号検索についても転送時のフィルタリングで対応できると考えられる。

3.5 キャッシュヒット時の性能

キャッシュがヒットした場合、問合せの処理時間は、理想的には問合せ結果の DMA 転送時間と等しくなる[*4]。ここでは、ハードウェア SQL キャッシュの動作速度を 200MHz とし、1 サイクル当たり 32-bit の DMA 転送ができると仮定する。**表 6** に、このときの OP.2〜OP.9 の転送データ量の期待値とその転送時間を示す。

3.6 キャッシュヒット率と性能

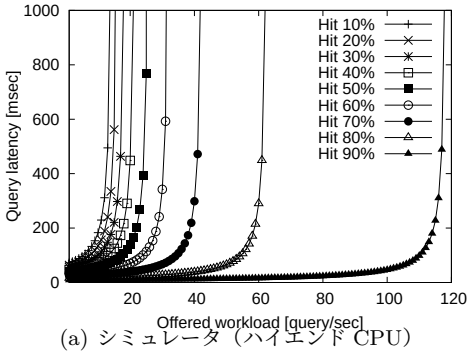
図 5 にキャッシュを用いた場合の問合せ処理のスループットを示す。スループットの計算には 2.4 節と同様に待ち行列理論を用いた。具体的には、表 4 及び表 6 のパラメータを用い、キャッシュヒット率を 10、20、30、40、50、60、70、80、90%と変化させながら問合せ処理に要す平均時間を見積もった。図 5 より、キャッシュヒット率によって処理スループットが大幅に変化することが分かった。

4. 予備評価

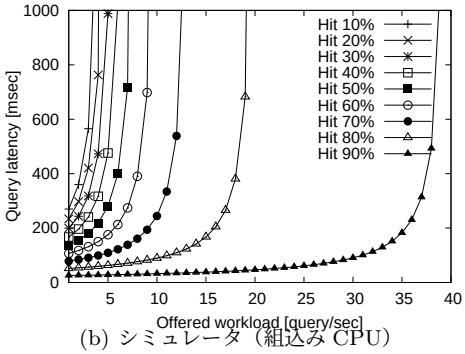
4.1 ハードウェア量に関する考察

ハードウェアによる SQL キャッシュの構造については、想定

[*4] 実際には、バスの調停にかかる遅延や DMA 転送の設定に要す遅延を考慮する必要がある。

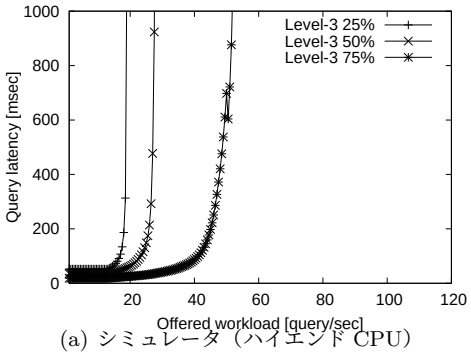


(a) シミュレータ（ハイエンド CPU）

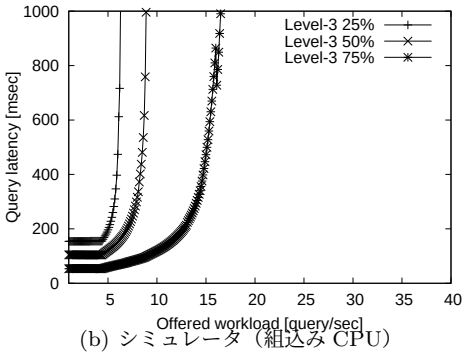


(b) シミュレータ（組込み CPU）

図 5 問合せ処理のスループット（CPU+提案キャッシュ 1 バンク、n=36,000、キャッシュヒット率は左から 10、20、30、40、50、60、70、80、90%の場合）



(a) シミュレータ（ハイエンド CPU）



(b) シミュレータ（組込み CPU）

図 6 ハードウェア SQL キャッシュのシミュレーション結果（n=36,000、全問合せのうち 25%、50%、75%が Class-3 問合せの場合）

するアプリケーション（データベースの利用法）に応じて変更が生じるため、FPGA に代表される再構成可能なハードウェアを用いて実現するのが適している。

本機構において最も物量を要するのは 3.4 節で述べたバッファキャッシュ（4,500kByte）の部分である。大規模な FPGA として Xilinx 社の Virtex-6 ファミリに注目すると、最大 38,304kbit の Block RAM を持つデバイスがあり [9]、本機構を十分に実装することができる。

さらに、Virtex-6 FPGA ML605 評価キット（デバイスは XC6VLX240T、Block RAM サイズは 14,976kbit）を対象に、Verilog HDL 及び Block RAM を用いて、結果キャッシュとバッファキャッシュ 2-Slot 分をマッピングした結果、十分に実装可能であることが確認された。

4.2 性能に関する考察

ここまでの性能に関する見積りは、すべて待ち行列モデル M/G/1 を用いて計算してきた（問合せ処理時間は表 4 にしたがう一般分布であると想定していた）。一方、本節では C 言語を用いて実装したハードウェア SQL キャッシュのシミュレータを用いてより正確な性能見積りを行う。

シミュレーションの入力として、ランダムに生成した 100,000 個の問合せミックスを用いた。この問合せミックスには、ランダムに生成した Class-1 問合せをベースに、高頻度問合せである Class-3 問合せを一定量（25%、50%、75%）混ぜた。これ以外の評価パラメータはこれまでと一緒である。

図 6 にシミュレーション結果を示す。当然、シミュレーション開始直後はすべての問合せがキャッシュミスするが、結果キャッシュ及びバッファキャッシュが埋まってくると、キャッシュがヒットするようになり性能が向上した。予想通り、性能向上率は問合せミックスに含まれる Class-3 問合せの割合に強く依存しているが、Class-3 が 25%のときで約 1.52〜1.94 倍の性能向上、50%のときで約 2.25〜2.78 倍の性能向上、75%のときで約 4.22〜5.12 倍の性能向上が確認された。

5. まとめと今後の課題

本論文では、サービス指向ルータにおける問合せ処理高速化のためにハードウェアによる SQL キャッシュを検討した。まず、プロセッサ単体の問合せ処理性能を見積もった。次に、ハードウェアによる SQL キャッシュのための設計方針を決め、問合せの多様性とカテゴリ分けを行った。これらの議論を踏まえ、ハードウェアによる SQL キャッシュとして、集合関数の結果をキャッシュする結果キャッシュ、及び、直近 1 時間の全レコードを保持するバッファキャッシュを設計し、待ち行列理論を用いて性能向上率を見積もった。最後に、ハードウェア量の考察を行い、また、C 言語で実装した SQL キャッシュシミュレータを用いて提案機構によりプロセッサ単体のときと比較して数倍の性能向上が得られることを確認した。

今後は、サービス指向ルータのアプリケーションを十分に考慮したうえでハードウェア SQL キャッシュの要件を詰めるとともに、今回設計したハードウェア SQL キャッシュの全機能を Verilog HDL を用いて実装し、FPGA 評価ボード上で動作させる。また、今回は主に選択演算に着目したが、結合や射影など他の主要な演算についても対応していく必要がある。

謝 辞 本研究の一部は、情報通信研究機構委託研究「新世代ネットワークを支えるネットワーク仮想化基板技術の研究開発」の支援による。

参考文献

[1] 西 宏章，松谷宏紀，久保亮吾，石田慎一，下條敏男，岩崎慶介，川島英之，藤原一毅，鯉渕道紘，田所将志：サービス指向ルータ・インフラストラクチャ，電子情報通信学会技術研究報告 IN2012-76， Vol. 112, No. 230, pp. 13–17 (2012).

[2] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The gem5 Simulator, *ACM SIGARCH Computer Architecture News*, Vol. 39, No. 2, pp. 1–7 (2011).

[3] 増永良文：リレーショナルデータベース入門，サイエンス社 (2003).

[4] SQLite: *http://www.sqlite.org*.

[5] Patterson, D. A. and Hennessy, J. L.: *Computer Organization and Design (4th ed.)*, Morgan Kaufmann (2011).

[6] Memcached: *http://memcached.org*.

[7] Little, J.: A Proof for the Queueing Formula:L=λW, *Operations Research*, Vol. 9, No. 3, pp. 383–387 (1961).

[8] MurmurHash: *http://sites.google.com/site/murmurhash/*.

[9] Xilinx: *Virtex-6 Family Overview DS150 (v2.4)* (2012).