Vol.2011-ARC-196 No.5
2011/7/27

オンルータデータベースにおける インデックス生成機構のマルチクエリ対応

西田雄介† 川島英之‡ 西宏章†

データベースマネジメントシステムをハードウェア化し，ネットワークストリームを直接処理することでネットワークデバイスがエンドホストに対してサービスとして提供できる可能性がある．本報告ではネットワークストリームをデータベース化することを想定したインデックス構造の検討を行い，マルチクエリ環境に対応したインデックス生成機構を FPGA 上に実装した．Xilinx FPGA Vertex5 を利用して論理実装した結果，パケットサイズが 50byte のときに 12.7[Gbps], 1036byte のときに 20.3[Gbps]のスループットを達成し, 数 Gbps 程度のネットワークで，ワイヤレートでの動作が可能であることを確認した．

Implementation of Index Generation Mechanism for Database on Router

YUSUKE NISHIDA† NOBUYUKI KAWASHIMA‡
HIROAKI NISHI†

By implementing DBMS as a hardware circuit, there is a possibility to provide a service to end-hosts by using network devices when the devices process network streams directly. In this paper, we propose the hardware structure of on-router database management system, especially its indexing mechanism. and the structure was implemented it by using FPGA. It proves that the throughput of the proposed architecture meets a wire-rate processing of network by fulfilling the throughput of 12.7Gbps in 50 byte packets and 20.3 Gbps in 1036 byte pakcets.

1. はじめに

近年，Google，Amazon，Flickr，YouTube など，API として無償で提供した web サービスを利用することで, 容易に高機能な web ページを作成することが可能となった．この背景には，マッシュアップなどインターネット上の複数発信源の情報を組み合わせる手法が登場し，より豊かなユーザエクスペリエンスをもたらすサービスが比較的容易に構築できるようになり，多角的価値のあるコンテンツを増産しているという状況がある．インターネットの発展は，より豊かなコンテンツやサービスに対する需要をもたらすと考えられるため，交換されるコンテンツの内部を直接扱うことでユーザの嗜好にあった，より高度かつ包括的なサービスを提供することが望まれている．

比較的大きな情報を交換する場合，複数のパケットに分断されて通信が行われる．これら分断されたパケット全体をストリームと呼ぶ．例えば，TCP/IP によって管理されたまとまりのあるデータにより作成されるパケット群を TCP ストリームと呼ぶのと同様，あるまとまりで意味を成すパケット群をネットワークにおけるストリーム，すなわちネットワークストリームと呼ぶ．ネットワークで交換されるパケットは，サービスから見れば情報の断片でしかないため，サービスの提供という観点では，ネットワークストリームという単位で情報を管理･解析することが必要である.すなわち，根とワークストリームの管理や解析には，ばらばらのパケットからストリームを再構築することが必要となる．ネットワークストリームという形で情報を利用するメリットは大きいが，未だエンドホスト以外，例えばゲートウエイやルータなどといった部位でのネットワークストリームの利用例は限定的である．これは，ストリーム再構築をハードウェア行うと，高い処理スループットが獲得できる可能性がある一方で，要求されるメモリ量が膨大となり実装が難しいという問題がある．菅原らは FPGA を用いて，ストリーム再構成を行うハードウェアを提案しており，必要なメモリ量の削減について議論している[1].

ネットワークストリームはインターネットユーザの行動履歴や，ストリームの出現時刻など，未だコンテンツとして利用されていない情報を多く含むため，これらの情報をコンテンツとして活用できれば従来に無いサービスを提供できる可能性がある．我々は，ネットワークストリーム情報が最も集中するハードウェアはネットワークルータであるとの考えから，コンテンツの扱いを可能とするサービス指向ルータを提案している．サービス指向ルータは，ワイヤレート処理を実現するため，データベースマネジメントシステムをハードウェア化し，ルータ上位でネットワークストリームを直接処理するとともに必要な情報を蓄積することで，通過するコンテンツをサービスへと転化する．ネットワークストリームをコンテンツとして利用するには，ストリームから必要なデータのみを抽出してデータベースへ格納することが求められる．すなわち，ストリームデータベースなどにより，セレクションを考慮してストリームインデックスを管理し，ストリームの中の必要な情報をルータ上のメモリやストレージにアーカイブする必要がある。

サービス指向ルータにおける懸案はデータベースへの書き込みおよびインデックス生成のスループットがワイヤレートを達成可能かどうかである．近年メモリへの書

† 慶應義塾大学
Keio University
‡ 筑波大学
Tsukuba University

き込みスループットは数十 Gbps 以上あるため，ワイヤレートを達成可能であるが，インデックス生成をソフトウェアで行った場合のスループットは数百 Mbps に留まる．また，このようなオンルータデータベースは巨大なメモリを搭載することが前提であるため，ストリーム再構築に専用メモリを搭載せず，ストア用メモリを再構築に併用することでメモリ利用効率とメモリ実装量の低減が期待できる§.

本稿ではネットワークストリームを再構築しつつインデックスをワイヤレートで生成し，特に複数クエリが混在する環境において，効率よく管理することを想定したインデックス生成機構をハードウェア実装することを目的とする．また，決定した構造に基づくインデックス生成機構を設計し，FPGA 上に実装したうえで，回路規模や動作速度といった評価を行う．

2. 関連研究

膨大なストリームデータに対し，外部記憶装置に蓄積することなくデータ検索を可能にする取り組みとしてストリームデータベースがある．ストリーム処理エンジンに関する研究はデータモデルの提案やメモリ使用量抑制の研究，分散処理化，具体的なアプリケーションを想定した専用システムの構築などが行われた[2][3][4]．ストリームデータ処理の具体例として，RFID やセンサなど，高いレートで生成される継続的データのリアルタイム処理が挙げられる．そのため，ストリームデータベースには高速処理が求められる．また，本報告の想定するネットワークストリームを有効なサービスとして利用する場合には，ワイヤレートで流れるネットワークストリームを継続的に取得・管理する必要がある．これらのことから，ストリームデータベースとネットワークストリームの有効利活用の実現には，データ挿入処理の高速化という共通目的が存在する．

これらのハードウェア化に関する研究もおこなわれているが，すべてセレクションにおけるデータベース内部基本操作のハードウェア化がメインであり，インサーションに関する研究が今後求められている．

3. インデックス生成機構の提案

ネットワークストリームをデータベースへ格納し，サービスとして提供するとき，ストリーム全体をストアせず，ユーザが指定した抽出項目に従って選択的に抽出しストアするほうが効率的であるため，本報告で到着したデータはクエリにより抽出済みであることを仮定する．リソースが限られることや，ストアの効率的な処理の観点からインデックス生成機構は以下の要求仕様を満たす必要がある．

§ アーカイビングはストレージなど不揮発なメモリへのストアに利用する用語であるため，揮発メモリへの書き込みを想定する場合，特に本稿ではストアと呼ぶ．

(1) 各ユーザが指定した抽出項目に従ってデータを管理し，読み出し時に，ユーザ単位の一括取得を可能とすることが可能なインデックス構造を管理すること
(2) ユーザ数の増大に対してある程度スケーラブルなインデックスを構成すること
(3) メモリリソースが限られることや，メモリスループットが限られることから，複数のクエリにより抽出されたデータは複製せず，インデックスで重複を管理すること

本報告ではこれらの要求仕様を満たし，ネットワークの状況及び発行されたクエリに応じてインデックス構造を適切に変化しながらインデックス生成を行うインデックス生成機構を提案する．本節ではインデック生成機構のアーキテクチャについて述べ，生成するインデックス構造について検討を行う．

3.1 アーキテクチャ

図 1 にインデックス生成機構のアーキテクチャを示す．Main Controller はインデックス生成機構の各モジュールへの命令信号を送信する．User-ID Table Controller は主に

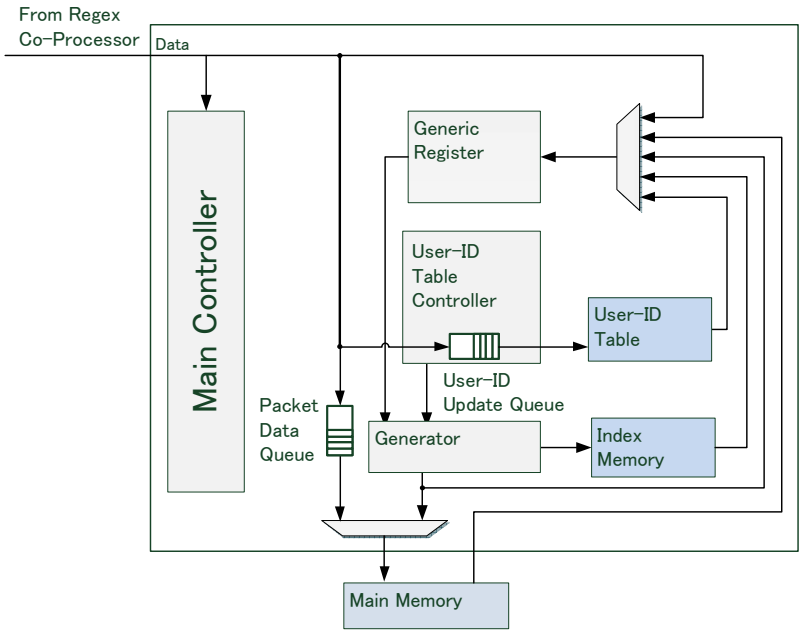


図 1：インデックス生成機構のアーキテクチャ

ユーザ ID の管理を行う．また，その他に User-ID Table からの読み出し処理や User-ID Table の更新処理を行う．Generator はインデックスの生成処理，書き込み処理，更新処理を行う．また Index Memory におけるフリーリストの管理を行う．Generic Register はインデック生成機構の各処理において必要となる情報を保持する．

3.2 インデックス構造の検討

本報告ではインデックス生成機構で生成するインデックス構造について以下の 3 つの手法を想定する．以下の UserList は所有者リストポインタを表し，同一のクエリにより抽出されたデータが格納されている次のセグメントのアドレスへのポインタである．抽出したデータが複数のクエリに重複してマッチすることを考慮したインデックス構造として，本報告では 3 つのインデックス構造についての検討を行う．

手法 1) クエリが重複したデータに対して，個別に ID を与えることで全体として重複することなく管理する．UserList ポインタは新たに付加した ID ごとに管理する．

手法 2) 一つのインデックスに対して複数の UserList ポインタを付加する．

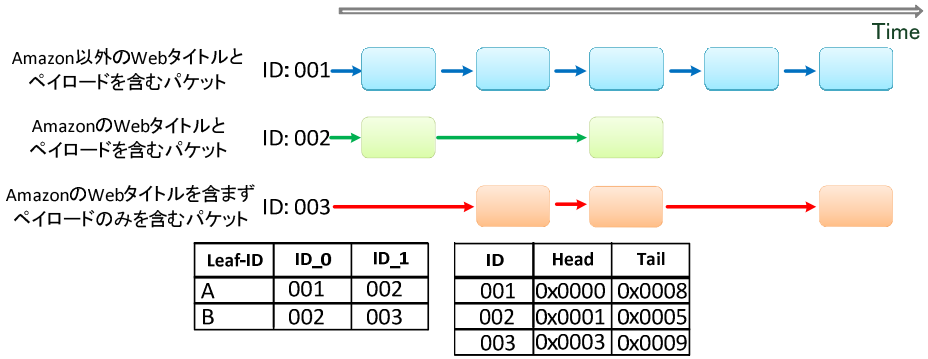
手法 3) メインメモリ上で抽出したデータとインデックスを管理する．複数のクエリが重複した場合はスタックとして UserList ポインタを管理する．

本節では上記の 3 つの手法について検討を行い，実装するインデックス構造の検討を行う．以下では 3 つの手法を検討するにあたり，ユーザ A が Web 閲覧におけるタイトルページとそのペイロード上位 500byte を，ユーザ B が Amazon のページにおけるペイロード 1kbyte を抽出するクエリを発行した場合を例に述べる．

3.2.1 手法 1 におけるインデックス構造

手法 1 では図 2 に示すように発行したクエリによって生じ得るすべての組み合わせについて，クエリ ID とは別の仮想クエリ ID を付加し，データを管理する．仮想クエリ ID ごとに保存領域を管理するため，ユニークな仮想クエリ ID 分の保存領域リストを保持する必要がある．なお，仮想クエリ ID 毎にリスト先頭アドレスと末尾アドレスを最低管理する必要であるが，仮想クエリ ID 数は発行クエリを n とするとワーストケースでは$O(2^n)$で付加すべき仮想クエリ ID 数が増大する．仮想クエリ ID の管理数が膨大となることが予想されるため，高スループットを達成するためにはクエリ ID をハッシュテーブルで管理するなどして，高速で参照する機構が必要である．一方で，この手法は生成するインデックス情報量が一定であるため，ハッシュテーブルへのアクセスを除けばインデックス生成スループットを一定かつ高い値で維持することが可能である．また，二つのテーブルを管理する必要があることや，新たにクエリが発行された場合にすでに発行されているクエリとの組み合わせを考慮して仮想クエリ ID 情報を更新しなければならないため，仮想クエリ ID の生成に伴う計算コストや，テーブル管理コストが高いといえる．

仮想クエリ ID を管理するテーブルは，発行されたクエリごとに全組み合わせをサポートする場合，その組み合わせ分のカラムを用意する必要があるためテーブルのメモリ使用効率が低下する．全組み合わせをサポートすることは現実的ではないため，図 3 のように管理テーブルを構築し，発行可能な仮想クエリ数を 65,536 と定めれば，クエリ ID は 16bit で表現でき，さらに，仮想クエリの重複を 1 つのエントリにつき 16 クエリまでと定めると，1 エントリの大きさは 34byte となる．仮想クエリ ID を管理するエントリにおいて，抽出データに対するクエリの重複数が増えると，仮想クエリ ID の登録数が増加する．一つのエントリにつき，管理する仮想クエリ数が 16 を超える場合はさらに別のエントリを獲得し格納する必要がある．図 4 に手法 2 におけるテーブルメモリ使用効率を示す．横軸は平均クエリ重複数を，縦軸はテーブルメモリの使用割合を表している．



Leaf-ID	ID_0	ID_1
A	001	002
B	002	003

ID	Head	Tail
001	0x0000	0x0008
002	0x0001	0x0005
003	0x0003	0x0009

図 2：手法 1 におけるインデックス構造

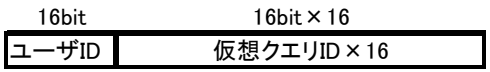


図 3：手法 1 における仮想クエリ ID 管理テーブルの構造

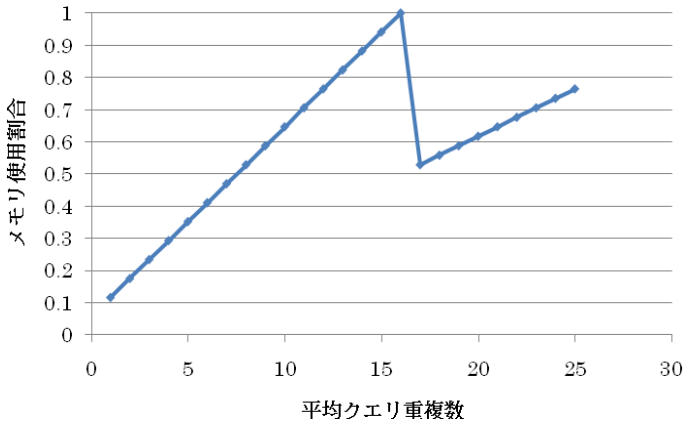


図 4：クエリ平均重複数に対するテーブルメモリ使用効率

3.2.2 手法 2 におけるインデックス構造

次に手法 2 について述べる．手法 2 においてはインデックス構造を図 5 に示すように，一つのインデックスに対して複数の UserList ポインタを付加する．この手法は，クエリを発行するユーザ数とテーブルで管理するユーザ数が一致するため，手法 1 よりもテーブル管理コストは低いといえる．一方で，一つのデータブロックに対して複数の UserList ポインタを付加させるため，インデックス管理が複雑となることや，抽出データのクエリ重複数に応じてインデックス生成速度が変化するために処理速度が低下することが想定される．また，この手法では全てのクエリについて UserList ポインタをサポートする場合，インデックスの一つのエントリにつきクエリ発行数分のカラムを用意する必要があるため，クエリの発行数が多い場合にはインデックスメモリのメモリ仕様割合が低下する．ここでは手法 1 同様に，クエリの最大重複数を 16 とした．図 6 に手法 2 におけるインデックスのエントリを示す．Time_next，Time_prv は時間リストポインタであり，Timestamp はデータの生成時刻，Number of Owners は抽出データにマッチしたクエリ数を表す．インデックスの一つのエントリのサイズは 80byte となる．この手法ではクエリ重複数が 16 を超えた場合についてもサポートする時はインデックス情報を別のメモリで管理する必要がある．図 7 に発行クエリ数とインデックスメモリ使用割合の関係を示す．横軸は平均クエリ重複数を，縦軸はインデックスメモリのエントリ使用割合を示している．

手法 1 と手法 2 では，クエリ数が少ない場合は手法 1 によるインデックス管理が有効であり，クエリ間での重複が多い場合は手法 2 によるインデックス管理が有効である．すなわちクエリ重複数とのトレードオフになると考えられる．

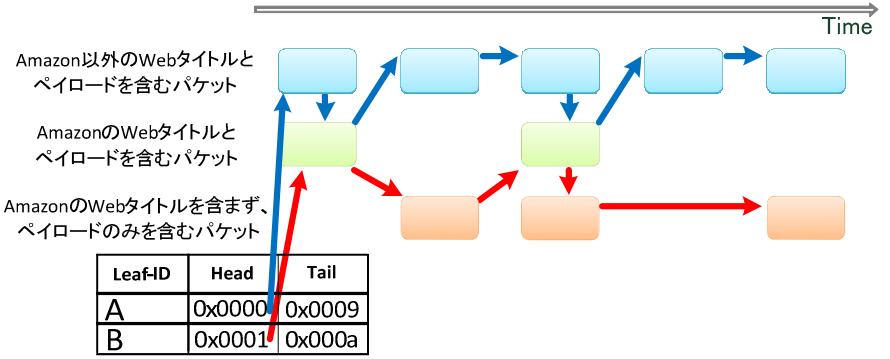


図 5：手法 2 におけるインデックス構造

32bit	32bit	32bit	32bit	32bit × 16
Time_next	Time_prv	Timestamp	Number of Owners	UserList × 16

図 6：手法 2 におけるインデックスのエントリ

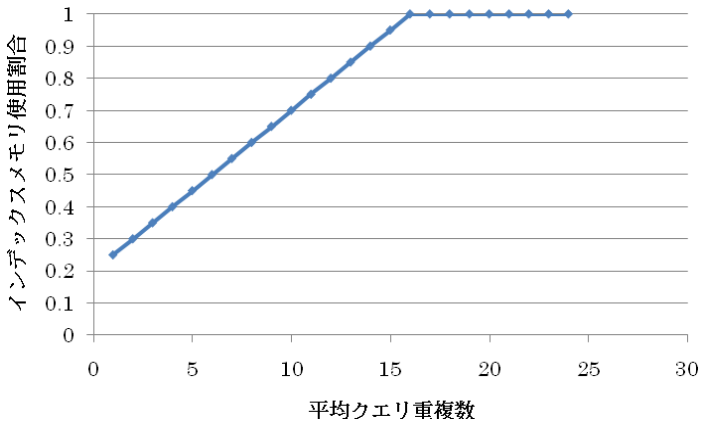


図 7：クエリの平均重複数に対するインデックスメモリ使用効率

3.2.3 手法 3 におけるインデックス構造

最後に手法 3 について述べる．手法 3 は手法 1，2 と異なり，メインメモリでインデックスを管理する．この手法では図 8 に示すように，セグメントのヘッダとして固定長のインデックス情報を格納する．一つのセグメントに複数のクエリにより取得されたデータが格納された場合は，インデックス情報内のクエリ ID を複数ユーザが所持していることを示すフラグを立て，各ユーザのインデックス情報をセグメントの末尾から格納する．この手法ではインデックス用のメモリやテーブル用のメモリを必要とせず，手法 1，手法 2 では発行されたクエリが増加するとテーブルメモリ，インデックスメモリの使用割合が低下するという問題が無くなることがメリットとして挙げられる．一方で，インデックスの書き込みもメインメモリにアクセスするために対応可能なスループットが低下することや，同様の理由により抽出でデータの読み出し処理速度が低下することが考えられる．

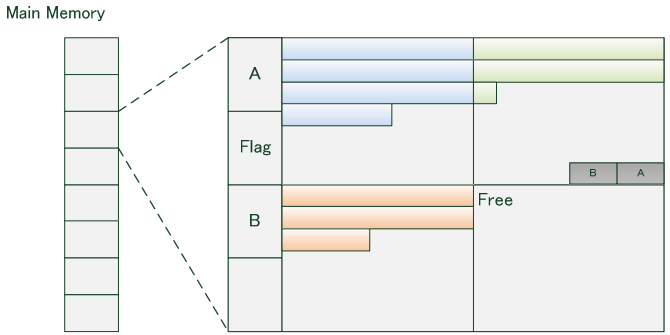


図 8：手法 3 におけるインデックス構造

以上の検討により，スループットにおいては手法 1，手法 2，手法 3 の順で有利であると考えられる．一方でクエリ数の増大に対するスケーラビリティは手法 3, 手法 2, 手法 1 の順で有利であると考えられる．これらのインデックス構造では抽出されたデータは UserList ポインタによりユーザ単位でのアクセスが可能であるため，要求仕様(1)を満たす．さらに，複数のクエリにマッチしたデータに対しても複製を行わないため要求仕様(3)も満たす．要求事項(2)については，各手法について比較を行う必要がある．本報告では初期評価として，手法 2 によるインデックス生成を行うインデックス生成機構を実装した．

3.3 設計

本報告では手法 2 によるインデックス生成機構を実装し評価する．手法 2 におけるインデックス管理情報として，抽出データの所有者数を示す共通管理数，時間リストによるアクセスを可能とするための時間リストポインタ，所有者リストポインタ，データのタイムスタンプを考慮する．以下にインデック生成機構の処理手順を示す．

1, ユーザ ID 情報と Timestamp をこの順に受け取り，インデックス生成を開始する．同時に Index Memory から Free Address を取得する．
2, 1 で生成したインデックスを, 取得した Free Address に書き込む. 取得し, Timestamp, 時間リストポインタ等の情報を，Index Memory へ書き込む．
3, ユーザ ID を用いてユーザテーブルへアクセスし, ユーザの末尾アドレスを取得する．
4, 3 で参照したユーザの末尾アドレスにおけるインデックスに到着したデータの書き込み先アドレスを所有者リストポインタとして記録することにより，UserList を更新する．
5, ユーザテーブルに再度アクセスし，そのユーザにおけるユーザテーブルの末尾アドレスの項目を更新する．
6, 3，4，5 を繰り返し，到着データに付加されたユーザ ID が無くなるまでインデックスの更新処理及び User Table の更新処理を続け，User ID が無くなり次第インデックス生成処理を終了する．

4. 実装・評価

3.2 項で設計したインデックス生成機構を Xilinx ISE Design Suite 12.3 で合成し，Xilinx の FPGA Vertex5 XC5VLX330T 上に実装した．また，FREEPDK45n テクノロジを用いて，Synopsys Design Compiler 2005.09 で論理合成した．また，それぞれの論理合成における回路規模，動作遅延，最大動作周波数を求めた．

表 1 にそれぞれのソフトにおける論理合成結果を示す．Xilinx ISE Design Suite により論理合成した結果，動作遅延は 3.73ns，最大動作周波数は 268MHz，使用セル数であらわされる回路規模は 337 個の LUT(Look Up Table)と 138 個の Register を使用する結果となった．また，Synopsys Design Compiler により論理合成した結果，動作遅延は 1.14ns，最大動作周波数は 877MHz，回路規模は $2.95 \times 10^3 \mu m^2$ となった．これらの数値より，各合成ツールで実装した場合のインデックス生成機構の対応可能式でスループットを算出した．スループットは(想定するパケットサイズ)／(インデックス生成処理時間)で表される．また, インデックス生成処理時間は(動作遅延)×(処理クロック数)で表される．表 2，3 に平均クエリ重複数が 1～4 のときにおける，パケットサイズが

表 1：インデックス生成機構の論理合成結果

	動作遅延 (ns)	最大動作周波数 (MHz)	回路規模	
Xilinx ISE Design Suite	3.73	268	LUT	Register
			337	138
Synopsys Design Compiler	1.14	877	$2.95\times10^3\ \mu m^2$	

50byte の時と 1306byte のときのインデックス成績校が対応可能なスループットを示す．なお，50byte はパケットサイズとして想定されるほぼ最少のサイズであり，1306byte は WIDE MAWI の 2009/7/12/14~14:15 の HTML パケットの平均サイズである．表 2，3 より平均クエリ重複数が 4 程度ならば，どちらの論理合成ツールで合成してもインデックス生成機構は数 Gbps 程度のスループットに対応できることがわかる．

表 2 ：FPGA を用いたインデックス生成機構の想定スループット

クエリ重複数	インデックス生成 クロック数	スループット(Gbps) 50byte	1306byte
1	6	13.4	17.1
2	7	13.4	17.1
3	10	10.7	17.1
4	13	8.25	17.1

表 3：ASIC を用いたインデックス生成機構の想定スループット

クエリ重複数	インデックス生成 クロック数	スループット(Gbps) 50byte	1306byte
1	6	43.9	55.9
2	7	43.9	55.9
3	10	35.1	55.9
4	13	27	55.9

また，Xilinx ISE Design Suite で合成したインデックス生成機構の対応可能なスループットの推移を想定するパケットサイズを 50byte から約 1500byte の範囲で算出した．図 9 に算出結果を示す． なお，1500byte はパケットサイズとして想定されるほぼ最大のサイズである． 算出した結果，パケットサイズが想定されるほぼ最小のサイズの 50byte のときでもクエリの重複数が 35 以下ならば対応可能なスループットは 1Gbps を上回ることが分かった．またパケットサイズが WIDE の HTML パケットにおける平均のサイズならば，クエリの重複数が 100 の場合でも 9Gbps のネットワークに耐えうることが分かった．

5. 結論・今後の課題

本報告は，ネットワークストリームをサービスとして利用することを想定し，データベースマネジメントシステムをハードウェア化することにより実現することを考えた．ストリームをサービスとして利用するには，データベースのインデックス構造の検討が不可欠でるため，本報告は実装するインデックス構造の検討を行い，それに基づくインデックス生成機構の設計を行った．

設計したインデックス生成機構の対応可能なスループットを調べた結果，Xilinx ISE Design Suite 12.3 により合成し，FPGA 上に実装した場合ではパケットサイズが 50byte

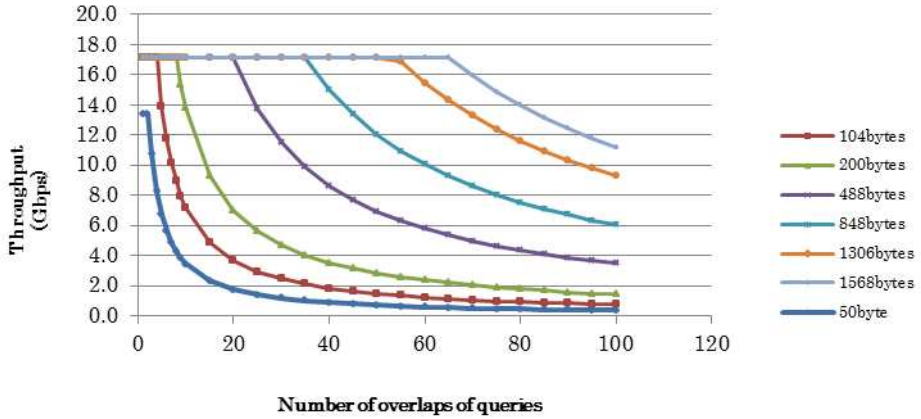


図 9：クエリ重複数に応じたインデックス生成速度のスループット

と 1306byte のそれぞれにおいて 12.7[Gbps]，20.3[Gbps]となり，FREEPDK45n テクノロジを利用した ASIC で実現した場合はそれぞれ 20.3[Gbps]，32.3[Gbps]となることを確認した．この結果，設計したインデックス生成機構が数 Gbps 程度のネットワークではワイヤレートで動作することが可能であることを確認した．

今後の課題として，手法 1，3 におけるインデックス生成機構の実装，各手法におけるメモリ使用効率やスループットの比較・検討，およびネットワークの状況に応じてインデックス生成手法を柔軟に変化させることのできる機構の実装が挙げられる．

謝辞 この研究は，独立行政法人情報通信研究機構「新世代ネットワーク技術戦略の実現に向けた萌芽的研究」および文部科学省科学技術研究費補助金基盤研究C「安心・安全な情報提供を可能とするインターネット基盤の構築に関する研究」の一環としてなされたものである．

参考文献

[1] 菅原豊，千本潤介，稲葉真理，平木敬. 10 ギガビットイーサネットを対象とした再構成型ネットワークプロセッサ. 第 1 回リコンフィギャラブルシステム研究会論文集. pp.249-256. 2003

[2] S. Chandrasekaran, O. Cooper, A. Deshpande, and M. J. telegraphcq continuous dataflow processing for an uncertain world. In SIGMOD, pp. 668–668, 2003.

[3] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. Spade: the system s declarative stream processing engine. In SIGMOD, pp. 1123–1134, 2008.

[4] L. Girod, Y. Mei, R. Newton, S. Rost, A. Thiagarajan, H. Bal akrishnan, and S. Madden. A regular expression processor embedded in service-friendly router for future internet. In Conference on Innovative Data Systems Research, 2007.