情報処理学会論文誌 Vol.57 No.8 1787–1799 (Aug. 2016)

多様なデータ構造を有する Key-Value Store アプライアンスの設計

徳差 雄太[1,a)] 松谷 宏紀[1,b)]

受付日 2015年11月19日, 採録日 2016年5月17日

概要：近年，センシング技術の発展とそのログデータの大容量化によって，IoT（Internet of Things）向けセンサデータ集約システムなど，組み込み用途においても大容量のデータを扱う必要が生じている．大容量のデータを蓄積および利活用するには Key-Value Store（KVS）のようなシンプルな構造のデータストアが向いており，そのなかでも最もシンプルな memcached プロトコルに特化した専用ハードウェアが研究されている．しかし，memcached では Value として単一の文字列しか格納できず，データベースとして利便性に欠ける．近年では，Value にデータ構造を持たせることができる KVS が登場しており，これ単体でデータベースとして利用できる．本論文ではこのようなデータ構造を有する KVS のハードウェアアプライアンスの設計を提案する．そこで，我々はデータ構造ごとに専用演算器を設計し，クロスバスイッチにそれぞれの専用演算器を接続し，マルチコアシステムとして設計する．それにより，用途に応じたデータ構造のコアをカスタマイズ可能であり，スループットに応じてそのコアの数を増やすことが可能である．本論文では，プロトタイプとして文字列型のデータ構造コアを実装し，FPGA で実機検証を行った．その結果，Virtex-5 では 11 コアまでスケールが可能であり，スループットとして 7〜9 コアで 10 Gbps ネットワークのラインレートを処理することが可能であることを示した．

キーワード：Key-value Store，FPGA

Design of Key-Value Store Appriance Having a Variety of Data Strcture

Yuta Toksashi[1,a)] Hiroki Matsutani[1,b)]

Received: November 19, 2015, Accepted: May 17, 2016

Abstract: Due to recent advances on sensing technology and Internet of Things (IoT) technology, sensor data aggregation systems are required to handle a large volume of data even in embedded system domains. Key-Value Stores (KVS) can be used to handle such a large volume of sensor data and dedicated KVS appliances that support memcached, which is the simplest form of KVS, have been reported. However, since only a single string is stored as a Value for a corresponding Key in memcached, it is widely used as cache systems rather than databases. Recently, another form of KVS that supports various data structures has been released and widely used as databases. In this paper, we propose a hardware design of KVS appliances that support various data structures. In our design, dedicated processing elements (PEs) are implemented for each data structure and they are interconnected via a crossbar switch as a multi-core KVS accelerator. The number and types of the PEs can be customized depending on the application. In this paper, as a prototype system, we implemented string type PEs on an FPGA board. The experiment results demonstrate that the number of PEs can be increased to 11 in a Virtex-5 FPGA. As KVS processing throughput, 7 to 9 PEs are required for 10 Gbit Ethernet line rate.

Keywords: Key-value Store, FPGA

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University, Yokohama, Kanagawa 223–8522, Japan
[a)] tokusasi@arc.ics.keio.ac.jp
[b)] matutani@arc.ics.keio.ac.jp

1. はじめに

近年，ネットワークインフラストラクチャの広帯域化やセンサデバイスの増加によって，大容量なデータ処理の必

要性が高まっている．とくに関係データベースと比較してスケールアウトに特化している点からNOSQL（Not Only SQL）が注目されており，多くのIoT向けのセンサデータ蓄積システムといった用途に利用されている．NOSQLに分類されるKey-Value Store（KVS）型はKeyとValueのペアを格納するデータストアであり，単純な構造であるため，多くのシステムのストレージやキャッシュなどの用途で利用されている．こうしたなか，大容量データ向けにKVSに特化したストレージ製品が導入されつつあり，このような製品では，データ処理性能に加えて，性能あたりの消費電力は重要な指標である．

KVSの中でもmemcached[7]をはじめとしてKVSのハードウェア高速化が近年さかんに研究されている[2], [3], [4], [5], [6], [8], [9], [10], [11], [12], [13], [14], [15], [16], [18], [21]．とくにField Programmable Gate Array（以下，FPGAと略す）を用いたハードウェアによる高速化はXilinx社などで行われている．とくにXilinx社は，memcachedのFPGAによるハードウェア化に取り組んでおり，10GbEのネットワーク環境でラインレートでの処理を達成している．文献[5]では，KVSのボトルネックとしてKVS処理の約60%がOSのネットワーク処理であると指摘している．処理に必要な演算が少ないという特徴からFPGAやASICによるハードウェア化が向いており，近年KVSのハードウェア化の研究が進んでいる．これらはmemcachedに代表されるようにKeyとペアとなるValueには任意の文字列や数値を格納する用途として設計されている．

memcachedに代表される多くのKVSでは，Keyごとにペアになる Valueとして単一の文字列を格納する．Keyと単一文字列から構成されるデータ構造はKVSをRDBMS（Relational Database Management System）のような外部データベースのキャッシュ層として利用するには十分であるが，IoT向けのセンサデータ蓄積システムとしてKVS自体をデータベースとして利用する用途には向かない．このような状況を背景に，データ構造を規定でき，単体でデータベースとして利用できるKVSが登場している．その代表例であるredis[20]では，任意の文字列をサポートする文字列型のほかにLIST型やHASH型，SET型，Sorted SET型など複数のデータ構造をサポートしており，近年，注目を浴びている[1]．しかし，KVSのハードウェア化については従来の文字列型KVSのみが報告されており，実用的な種々のデータ構造に対応したKVSのハードウェア化については検討されていない．本研究の目的は，実用的な種々のデータ構造に対応したKVSをいかにハードウェア化すればよいかを探求することにある．

それぞれのデータ構造を処理するデータベースコアを設計し，RTLシミュレーションを行ったところ，データ構造ごとに実行サイクル数が大きく異なるうえにHashTableに対する操作やステップ数も異なることが分かった．そのため，様々なデータ構造を単一パイプラインで処理するのではなく，データ構造ごとに専用Processing Element（PE）を複数設け，それらをクロスバ接続するマルチコアアーキテクチャを提案する．

本論文では文字列型，LIST型，HASH型，SET型のそれぞれのデータ構造ごとにPEを設計した．そして，プロトタイプ実装として文字列型PEをFPGAに実装し，実機での動作確認を行った．また，PEをマルチコア化した場合のシミュレーションを行った．その結果，10 Gbpsネットワークのラインレートを処理するためにGETリクエストでは7コア，SETリクエストでは9コアが必要となることが分かった．また，面積評価を行ったところ，Virtex-5 XC5VTX240TのFPGAに11コアまで搭載することが可能であることを示した．システム全体の消費電力は約30 Wであった．

本論文の構成は以下のとおりである．2章では，本研究の関連研究を示す．3章では，本論文で提案するヘテロジニアス・マルチコア方式を採用した専用PEの設計を論じる．4章では，専用PEの実装を述べる．5章では，本システムを用いたケーススタディを紹介する．6章では，その性能評価を行う．7章で本論文をまとめる．

2. 関連研究

KVSの高速化がさかんに行われている．文献[5]では，memcachedのボトルネックはOSのTCP/UDPプロトコル・スタックであると指摘している．そのため，ボトルネックとなるネットワーク処理やデータベース処理をFPGAで設計することで，ネットワークインタフェースの帯域速度でデータベース処理を高速化できる．とくに頻出するGET，SETリクエストをFPGAで実装している．また専用ハードウェアを使用しているため，CPUを使用したシステムと比較して消費電力を改善できる．このようにFPGAといった専用ハードウェアを使用することで，データベースの処理性能だけでなく，電力あたりの性能も改善できる．したがって，本研究で扱うFPGAを用いたKVSの設計も同様に電力効率の改善が期待できる．

Xilinx社では，FPGAによるmemcachedの高速化に取り組んでおり，FPGAのスタンドアロン設計はCPUとFPGAの協調設計と比較して，単位電力あたりのスループットを2.3〜6.1倍改善している[2], [4]．とくにワークロードで頻出するGETリクエストをFPGA上にオフロードし，処理を数100段に及ぶ深いパイプラインとして実装することで10 Gbpsネットワークのラインレートを実現できたと報告している．その一方で，複数のデータ構造を扱う場合は，データ構造ごとにデータベース処理の実行時間が異なる．また，新たなデータ構造を追加することも考慮に含めるとパイプライン化は向いていない．そこで，本研

究ではデータベース処理を行う複数の PE を FPGA 上に配置することで高速化を目指す．

また，文献 [15] では，FPGA ではなく，専用 ASIC として KVS アクセラレータを設計している．FPGA でのプロトタイプにおいて電力効率が 6〜16 倍改善されたと報告している．FPGA のような LUT の組合せではなく，専用 ASIC として実装することでさらなる性能の改善が期待される．本研究では，様々なデータ構造に対応した専用ハードウェアをマルチコア化することで高速化に取り組む．ASIC ではワークロードに応じてコアを組み換えるようなカスタム化が困難であるが，この欠点を克服するために本論文では FPGA を対象に設計を行う．

RAMCloud プロジェクト [18] では，ディスクアクセスの遅延時間が原因で，キャッシュミスにより 10 倍近くの速度の低下を引き起こすと指摘している．したがって，データベースのストレージとしてディスクを使用せず，DRAM などのメモリを用いることで高速なデータベース・アクセスを実現している．

また，汎用 CPU を用いた KVS の高速化も取り組まれている [14]．これはボトルネックになっているネットワークプロトコル・スタックをバイパスし，NIC（Network Interface Card）の Direct Acess を利用し，NIC の Receive Side Scaling の機能を用いて，複数の CPU にクエリ処理を分散することで高いスループットを実現している．しかし，汎用 CPU を用いており，電力効率を大きく改善することは困難であると考えられる．一方，本研究のようにネットワーク I/O やメモリ I/O に直接組み込みが可能なハードウェアを設計することはスループット向上と電力効率改善において効果的であると考えられる．

3. データ構造サーバの提案

本研究では，データ構造に対応した KVS について検討を行う．redis [20] に代表されるデータ構造サーバでは，Key と Value のペアとなる Value にデータ構造を定義することが可能である．redis では，以下に示すデータ構造に対応している．

- 文字列型：文字列型は任意のデータを Value として保持できる．WRITE 操作として SET（key，value），READ 操作として GET（key）といった API が提供されている．
- HASH 型：文字列型の順番のない Key とペアとなる Value にフィールドと Value の組合せを複数登録できる．chunk を使用して新しいハッシュテーブルを作成し，Value を登録する．WRITE 操作として HSET（key，field，value），READ 操作として HGET（key，field）といった API を提供している．
- SET 型：Key とペアとなる Value に集合を設定できる．ある程度大きいサイズの chunk を用意して，そこに集合の要素を一定の間隔で格納していく．WRITE 操作として SADD（key，member）命令，READ 命令として SPOP（key）といった API を提供している．
- LIST 型：文字列型をベースとした双方向のリスト構造である．新しい要素をリストの先頭や末尾に追加することができる．WRITE 操作としてリストの末端にデータを挿入する LPUSH（key，value），RPUSH（key，value），READ 操作として LPOP（key），RPOP（key）といった API を提供している．
- SORTED SET 型：SET 型と類似しているが，集合のそれぞれの要素はそれぞれスコアに関連したハッシュ値を持つ．要素はスコアに基づいて順番に整列される．WRITE 操作として ZADD（key，score，member）などの API を提供している．

表 1 データ構造型ごとの PE の実行クロックサイクル数
Table 1 Required clock cycles for each data structure PE type.

データ構造	READ	WRITE
文字列型	78	269
LIST 型	87	272
SET 型	87	477
HASH 型	78	269

それぞれのデータ構造型を処理する専用 PE を実装し，RTL シミュレーションを行った結果を**表 1** に示す（実装の詳細については 4.3 節で述べる）．文字列型をはじめ WRITE 操作ではメモリアロケーションを行っているため，READ 操作と比較して実行サイクル数が多くなっている．LIST 型は WRITE 操作として LPUSH 命令，READ 操作として LPOP 命令を想定している．文字列型と異なり，Key のハッシュ値をもとに参照する HashTable には LIST の先頭の Value アドレスと末尾の Value アドレスを格納している．LPOP では，値の参照に加えて，Value 領域の削除と LIST の次アドレスを書き換える操作が追加されるため，クロックサイクル数が文字列型よりも多い．

SET 型では，Value に値の集合を指定することが可能である．そのため，Value 管理に member 表を設けている．WRITE 操作において他のデータ構造と比較して，メモリアロケーションの領域が大きいため，他のデータ構造よりも実行サイクル数が多い．

HASH 型では，Key とペアとなる Value に，Field と Value を直結したものを複数指定することが可能である．ハードウェア設計にあたり，Key と Field を直結したものを Key とすることで，文字列型と等価と考えることができる．したがって，表 1 の HASH 型の実行サイクル数は文字列型と同値となる．

以上の結果から，複数のデータ構造や今後新しいデータ構造型に対応しようとすると単一のパイプライン実装は拡張が容易ではない．そこで，本論文では複数のデータ構造

表 2 実装対象の FPGA ボード
Table 2 Target FPGA board.

ボード	NetFPGA-10G
FPGA	Virtex-5 XC5VTX240T
Memory Type	RLDRAM-II
Memory Capacity	288 MB
SRAM	QDRII SRAM
SRAM Capacity	27 MB
PCIe	PCIe Gen2 x8
Network I/O	10GbE (SFP+) 4 本

に対応するヘテロジニアスマルチコア方式のハードウェア設計を提案する．データ構造ごとに専用 PE を設計し，クロスバスイッチに接続する．データ構造ごとの命令に応じてそれぞれの専用 PE に処理を渡すことが可能である．パイプライン化と異なり，PE 内でデータベース処理が完結するため，マルチコア化することで高速化を目指す．

4. FPGA ボードへの実装

本章では，データ構造を有するハードウェアによる演算器のプロトタイプとして文字列型 PE を市販の FPGA ボードに実装した．他のデータ構造用 PE については RTL 設計まで行い，RTL シミュレーションにより性能を評価している．

4.1 実装環境

実装では**表 2** に示すとおり 10GbE NIC を有する FPGA ボードを対象とした．NetFPGA Project [17] で提供されている Reference NIC の設計をもとにしている．RTL 設計には Verilog HDL を使用し，シミュレーション環境には Icarus Verilog を使用している．論理合成ツールとして Xilinx 社の ISE 13.4 を使用している．

4.2 KVS アプライアンス全体の構成

本実装では，ハードウェア設計のコストを考慮にいれ，通信には UDP/IP を用いているが，将来的に TCP/IP を用いることも考えている．

NIC の設計を含む全体のハードウェア構成を**図 1** に示す．Network Interface から入力されたパケットは PE Affinity モジュールで解析される．宛先 IP アドレスおよびサービス（UDP ポート）番号による論理積で一致したパケットを KVS クエリとして，NIC 内の各専用 PE で処理を行う．

PE Affinity でマッチしたパケットは，その後，クロスバスイッチに送られる．このクロスバスイッチはデータ幅が 128 bit 幅であり，複数の PE と SRAM コントローラ，DRAM コントローラが接続されている．クロスバスイッチはクエリのデータ構造タイプをもとにパケットを各 PE に分配する．各 PE はクエリの応答を生成する．各 PE の

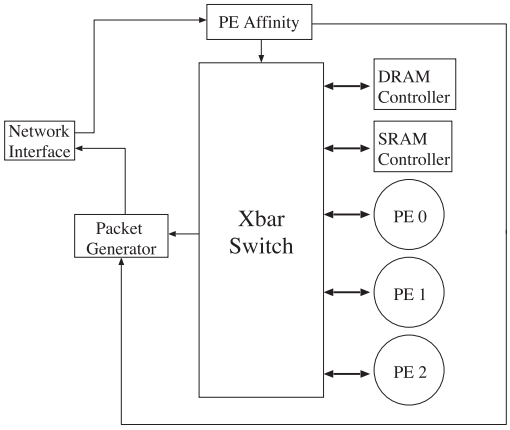

図 1 NIC を含むハードウェア全体図
Fig. 1 Hardware overview including NIC.

出力はクロスバスイッチでアービトレーションされ Packet Generator によって応答パケットが生成される．

4.3 データベースコアの設計

図 2 に例として文字列型 PE のブロックダイアグラムを示す．他のデータ構造については文字列型 PE の設計をベースとしている．HashTable のエントリの使用法や chunk のデータやメモリアクセスの回数が異なるが，文字列型 PE のアーキテクチャと大差はない．以下に文字列型 PE の相違点をまとめる．

- LIST 型：図 3 中の HashTable のエントリの Value Pointer と Reserve を組み合わせて，Right アドレスと Left アドレスの 2 つのアドレスを管理する．LPOP や LPUSH は Left アドレスからデータを追加したり，削除したりする際に用いる．RPOP や RPUSH は同様に Right アドレスを使用する．Value データを格納する chunk の先頭でも Right アドレスと Left アドレスを持っており，このアドレスを使用して，LIST 構造を実現する．この際，アドレスに使用する bit サイズ分だけ，オーバヘッドが発生する．
- HASH 型：クエリには, Key と Field, Value が含まれている．この Key と Field を直結したものを HashTable で管理する．Key と Field のサイズが 64B を超える場合，HashTable において複数エントリにまたがって格納される．
- SET 型：Key とペアとなる Value は member 表で管理されている．したがって，Key をもとに HashTable で探索されたアドレスの chunk には member 表が含まれている．この member 表で管理されている chunk にアクセスすることで member に含まれる Value にアクセスすることができる．文字列型に比べて，メモリアクセスの回数が多い．

クロスバスイッチから入力されるクエリは，Fetch モジュールでクエリ解析が行われ，Key と Value，Operation

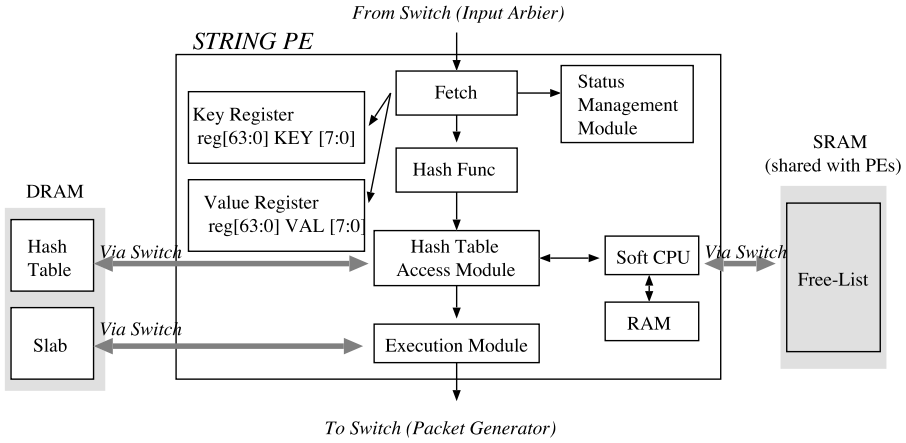

図 2 PE のブロックダイアグラム
Fig. 2 PE block diagram.

Code（文字列型であれば SET，GET）などを識別する．

Hash Function では Key をハッシュ関数 CRC32 を用いて 32 bit のハッシュ値を算出している．CRC32 を用いることで 1 クロックサイクルのレイテンシでデータパス幅分のデータをハッシュ値に変換できる．このハッシュ値をもとに HashTable を探索し，Value が格納されている chunk アドレスの取得や Value の更新を行う．Hash Table Access Module では DRAM 上にマッピングされている HashTable をルックアップする回路が含まれている（詳細は 4.4 節で説明する）．

4.4 スラブアロケータによるメモリ管理

DRAM には，KeyValue のペアを格納するストア領域への HashTable と実際の DataStore 領域の 2 つの領域がある．**図 3** に FPGA ボード上の DRAM 上に実現される DRAM 領域のレイアウトを示す．図中では NetFPGA-10G での例を示しており，搭載されている RLDRAM-II（288 MB）を対象としている．

HashTable 領域では 1 つのエントリが 72B である．これは，KeyValue のペアとなる Key を Entry の最後尾の Key に格納し，該当する Value の格納場所を先頭の Value Pointer で表している．クエリを処理する場合，この Key を比較し，一致した場合に各専用 PE は Value Pointer を参照する．

DataStore 領域では，スラブアロケータを用いてメモリ管理を行う．スラブアロケータの実現方法として CPU を用いたソフトウェア処理が考えられる．KeyValue の Write クエリを受信すると，この Key と Value を格納するために chunk と呼ばれるメモリ領域を DRAM 上に新規に割り当てる．その際，ソフト CPU を用いて，SRAM に格納されている FreeList の情報をもとに，利用可能な DRAM の chunk を割り当てる．PE の処理系は Write 時のメモリ割当て処理を待つ必要があるため，ソフト CPU の処理速度がボトルネックになる可能性がある．ここでは 1) 実装のシンプル化，2) スラブ構成変更の容易さ，および 3) ハード CPU を有する FPGA デバイスの普及の 3 つを考慮に入れ，スラブアロケータの処理はソフト CPU 上で動作するソフトウェアとして実現する．この利点を以下にまとめる．

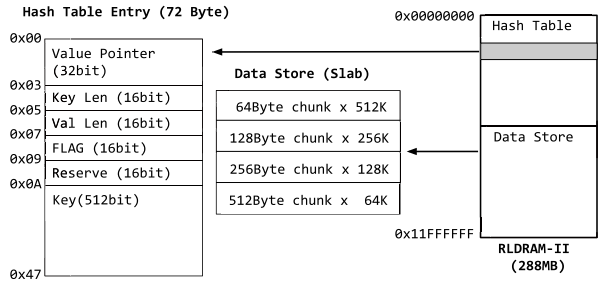

図 3 DRAM 領域のレイアウト（HashTable 領域および DataStore 領域）
Fig. 3 Memory space layout of on-board DRAM (HashTable and DataStore spaces).

1) 実装のシンプル化：可変長の Value に対応可能なスラブアロケータでは，サイズごとに chunk と呼ばれる領域を確保しており，利用可能な chunk のメモリアドレスをリスト管理する必要がある．これはポインタをともなうリスト処理であり，並列化は容易ではなく，専用ロジック化の恩恵が見込めない．そのためソフトウェア処理が向いている．

2) スラブ構成変更の容易さ：スラブアロケータでは，FPGA ボード上の DRAM 空間を 64B，128B，256B，512B などのサイズの chunk に区切って，利用可能なメモリプールを用意する．ここで，準備する chunk サイズおよびその個数は重要なパラメータであり，ワークロード（要求される Value のサイズ分布）に応じて柔軟に決定する必要がある．スラブアロケータをソフトウェアで実現することで，たとえば，64B の chunk が枯渇してきたら 128B の chunk の一部を分割して 64B の chunk を増やすなど，柔軟なスラブの動的再構成が可能となる．一方，スラブアロケータを専用ロジック化すると，ワークロードに応じて柔軟，か

表 3 スラブアロケータ用ソフト CPU の仕様
Table 3 Soft-macro CPU specification for slab allocator.

CPU	MIPS R3000 互換
命令・データ幅	32 bit
RAM	32 kB（キャッシュなし）
動作周波数	80 MHz

つ，動的に chunk サイズおよびその個数を変更することは困難になる．

3) ハード CPU を有する FPGA デバイスの普及：ソフト CPU は FPGA の動作周波数に依存し，ハードマクロによる CPU と比較すると低い周波数で動作する．今後，ソフト CPU をハード CPU に置き換えることにより，動作周波数の問題は改善されると考えられる．実際，Xilinx 社には FPGA 内に ARM のハード CPU を含む Zynq シリーズ，Altera 社にも同様に Cyclone というデバイスがある．本提案手法においても，今後は FPGA 内蔵のハード CPU を積極的に活用することで現在のソフト CPU の処理内容を高速化できる．

4.5 スラブアロケータの実装

可変長の KeyValue サイズに対応するために，本実装ではスラブアロケータを FPGA 内のソフト CPU に任せる．本システムではソフト CPU を**表 3** に示すように MIPS R3000 互換のプロセッサを各専用 PE に実装している．

また，事前にワークロードが特定できる場合，それに限定した Key 長，Value 長に特化したスラブアロケータ回路を実装することも可能である．ここでは一般的に可変長を扱えるようにソフト CPU を用いている．また，前節で述べたように FPGA の中には CPU のハードマクロを搭載しているものがあり，本設計のスラブアロケータをハードマクロ CPU 上で動作させることでソフト CPU よりも高い周波数で動作させることも可能である．今回対象とした NetFPGA-10G の Virtex-5 には CPU のハードマクロは含まれていないが，今後は Zynq など強力なハードマクロ CPU を有するデバイスが普及していくと考えられる．ハードマクロによる CPU コアを利用することでスラブアロケータの性能を改善できると期待されるが，今回は使用する FPGA の制約上，スラブアロケータ処理はソフト CPU で動作させる．

4.6 クロスバスイッチ

本システムでは各 PE はクロスバスイッチに接続されており，マルチコア化することで高速化を実現する．**表 4** に本システムで使用するクロスバスイッチの仕様を示す．データ幅は 128 bit 幅で，アービトレーションポリシは固定優先型を使用している．また，各 PE と同様にクロスバスイッチの目標動作周波数は 160 MHz である．

表 4 クロスバスイッチの仕様
Table 4 Crossbar switch specification.

データ幅	128 bit
アービトレーション	固定優先型
動作周波数	160 MHz

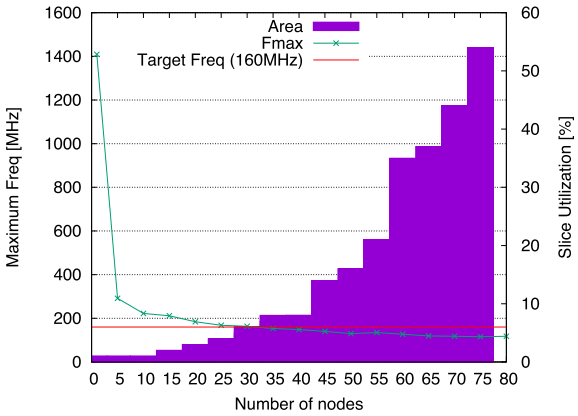

図 4 クロスバスイッチの PE 数におけるスライスの割合と周波数
Fig. 4 Maximum operating frequency and slice utilization vs. number of PEs connected to crossbar switch.

図 4 に NetFPGA-10G に搭載されている FPGA である Virtex-5 XC5VTX240T でクロスバスイッチのみを論理合成した場合の動作周波数およびスライスの占有率を示す．各 PE は 160 MHz での動作を前提としており，図中の赤線に示すように目標動作周波数（Target Freq）として 160 MHz を示している．クロスバスイッチの PE 数が 30 より多い場合，動作周波数（Fmax）は 160 MHz を下回ることが分かった．また動作周波数内でのスライスの占有率は 8%未満である．したがって，Virtex-5 のデバイスにおけるクロスバスイッチの面積および動作周波数の面では PE を 30 個まで接続できる．

5. ケーススタディ

本提案のハードウェアによる高速性と省電力性を活かすことができるアプリケーション例として，ここでは「行列待ち情報提供システム」への応用を考える．行列待ち情報システムでは，行列を撮影しているカメラの映像を画像解析することで性別や推定年齢を割り出し，行列待ちのユーザにマーケティング情報やコンテンツを提供する．このようなシステムはレストランやバス停といった小規模な用途から，遊園地や空港，大型ショッピングセンタといった大規模な用途までが考えられる．カメラで一度に撮影できる範囲は数十人分程度と考えると大規模な用途では多数のカメラによる運用が必要となる．このアプリケーションでは，不特定多数の人数のデータを管理しており，現在並んでいるユーザを管理するために，高スループットが要求される．また，電気代などの管理コストは小さければ小さいほど好ましいため，省電力性も重要な要素の 1 つであり，

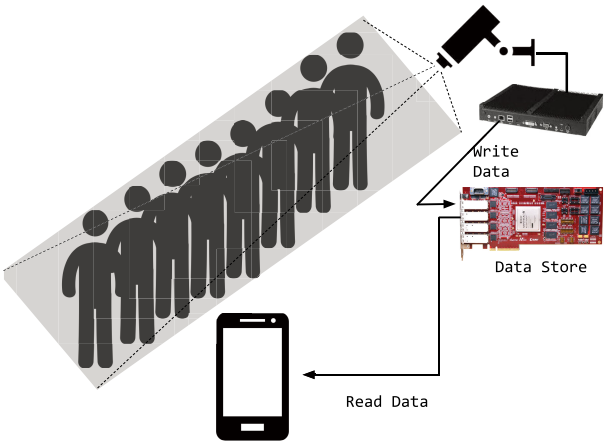

図 5 アプリケーション想定図
Fig. 5 Application overview.

省電力性を目指す本提案システムが適している．

5.1 システム構成

画像解析サーバとして沖電気工業株式会社の RESCAT CA [22] が配備され，個人特定を可能にするために各個人は RFID タグを所持していると想定する．**図 5** に本提案システムを活用したケーススタディの一例を示す．本システムはセンサデータ集約システムとして動作し，各ネットワークカメラで画像解析によって年齢や性別，滞在時間などを個別 ID ごとに取得できる．本システムはそのデータを管理するデータベースとして使用する．図中には複数あるネットワークカメラの 1 つが，ある待ち行列（レストラン，遊園地，バス停など）を撮影している．画像解析サーバはその解析結果を本システムのデータベースに格納する．そのため，データベースシステムには行列に並んでいる人数に相当する WRITE クエリ数を処理できる性能が求められる．

ユーザはスマートデバイスなどの移動体通信端末を用いて現在の待ち情報システムを確認したり，運営側から連絡を受信したりすることが可能である．ユーザや運営側はこれらの結果をデータベースから READ してきたデータをもとにスマートデバイス上に Web コンテンツとして表示する．したがって，不特定多数の利用者からの大量の READ リクエストの要求を処理できるだけの性能が必要となる．

5.2 データ構造および表現力の比較

表 5 に本システムを用いることで表現可能な LIST 型のデータ表現，**表 6** に単純な KVS をの場合（リストを表現できない場合）における同等のデータ表現の例を示す．表中の括弧内は変数，ダブルクオーテーション内は固定値とする．Value に LIST 型データ構造を定義できる場合，その Value はログのように履歴を管理することが可能である．一方で単純な KVS で同等の表現をする場合，ある時刻に

表 5 LIST 型 PE によるデータ構造の表現
Table 5 Data structure representation by List type PE.

Key	Value
{Current Time}+“gender”	“female”, “male”, “female”, ...
{Current Time}+“time”	“201502141311”, ...
{Current Time}+“age”	“23”, “45”, “48”, ...
{Current Time}+“location”	“35.55518314,139.65525201”, ...
{Current Time}+“rfid_tag”	“00000001”, ...

表 6 単純な KVS による表現（Index はリスト中の個々のデータの通し番号とする）
Table 6 Data structure representation by simple KVS (Index is used as serial number for each data on list).

Key	Value
{Current Time}+{UserID}+“gender”+{Index}	“female”
{Current Time}+{UserID}+“time”+{Index}	“201502141311”
{Current Time}+{UserID}+“age”+{Index}	“23”
{Current Time}+{UserID}+“location”+{Index}	“35.55518314, 139.65525201”
{Current Time}+{UserID}+“rfid_tag”+{Index}	“00000001”

取得された複数のデータ（行列中の複数人のデータ）から個々のデータを取得しようとすると，各データに {Index} を割り振り，{Index} を用いて個々のデータを指定する必要が生じる．

サービスの運用上，過去のデータを保持する必要がないため，サービス提供中のデータを保存するのに必要なデータ容量を算出する．ユーザ側のアプリケーションで処理しやすいように Value はバイナリではなく，テキストデータとして格納する．データは表に示す 5 種類で，Value はすべて 64B 未満で済むため，DataStore の構成として 64B の chunk のみを用いて運用する．1 人あたりの情報量は $64\text{B} \times 5 = 320\text{B}$ である．使用する Key はすべて 64B で格納できるものとする．DRAM で使用する HashTable と DataStore の容量比を 1:1 と想定すると，NetFPGA-10G では DataStore のサイズは 144 MB となる．ここで，chunk サイズは 64B の 1 種類で運用すると想定すると，471,859 人分のデータを格納できる．次世代ボードである NetFPGA-SUME の場合には DDR3 SDRAM の容量が 8 GB あるため，DataStore として 4 GB を使用でき，13,421,772 人分のデータを格納できる．したがって，行列に現在並んでいる人々のデータの管理用途であれば，FPGA ボード上に搭載される DRAM の容量でも十分に運用可能であると考えられる．DRAM の容量が不足する場合にはカメラを設置するエリアを分割して運用するなど分散して使用することも考えられるが，ここでは DataStore を分割しない環境を想定する．

表 7 クエリ処理時間 [単位：nsec]
Table 7 Query processing time [unit: nsec].

	LIST PE	memcached	redis
READ 操作	832.0	1647.5	5487.9
WRITE 操作	650.0	2416.6	5690.9
平均	741.0	2023.0	5587.6
分散	8281.0	147861.7	10305.6

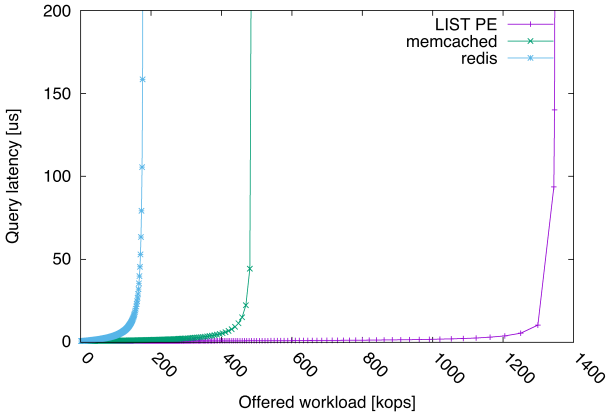

図 6 待ち行列モデルを用いたスループット比較
Fig. 6 Throughput comparisons based on queuing model.

5.3 データ量の評価

ここでは待ち行列モデルを用いて「行列待ち情報提供システム」で発生する KVS 処理のスループットを計算する．待ち行列モデルは M/G/1 を使用した．これは，行列に並ぶ人の到着率 λ はポアソン過程に従い，サービス時間は，**表 7** で示す平均と分散に従う一般分布，窓口は 1 つのモデルと考えられる．LIST PE と memcached，redis [*1] の処理時間を表 7 にまとめた．LIST 型 PE と redis は READ および WRITE 操作として LPOP と LPUSH クエリのそれぞれの処理時間，memcached は SET および GET クエリの処理時間を示している．これらによりサービス変動係数 $c = \sigma/\bar{x}$ を求める．これらをもとに Little の公式を用いてクエリを処理する待ち時間を計算する．

図 6 に本提案手法の 1 つである LIST 型のデータ構造の PE と memcached，redis の待ち行列で発生するスループットを示す．LIST 型 PE は約 1340 kops（operation per second）までのワークロードであれば約 50 us での待ち時間で処理できている．グラフでは単純な KVS として memcached(1thread) および redis(1thread) を使用した場合に行列待ちアプリケーションで処理できるスループットを示しており，memcached は約 100 kops，redis では約 450 kops であった．redis では，ソフトウェアでデータ構造を扱うための処理オーバヘッドを含んでおり，memcached よりも低速であることが分かる．memcached は LIST のデータ構造に対応していないため，複数のデータを取得するには Index の値をインクリメントしながら複数回 KeyValue ペアを取得する必要がある．そのため，LIST に対応している KVS に比較して必要なクエリ数が増える．以上の結果から，単純な KVS や LIST 型のデータ構造に対応しているソフトウェアによる redis と比較して，本提案の LIST 型の 1PE の処理性能は redis の 13 倍，memcached の 3 倍程度の処理性能を達成している．

[*1] LIST 型の LPOP，LPUSH による性能．

6. 評価

6.1 計測環境

表 8 の環境でスループット性能や消費電力の計測を行った．クライアント側では netmap framework [19] を用いた pkt-gen を活用して，10GbE ネットワークにおけるラインレートでパケットを生成している．GET と SET の Request パケットとして，それぞれ固定長の Key サイズ，Value サイズを想定している．また，比較対象として redis [20]，memcached [7] を用いた．2 章で説明したように，KVS のボトルネックはカーネルにおけるネットワーク・プロトコルスタックにある．それを改善するために，既存手法として，1) ネットワーク・プロトコルスタックをバイパスする手法，2) ネットワークインタフェースのデバイスドライバレベルで KVS 処理を行う手法の 2 点が考えられる．カーネルをバイパスする手法として netmap framework [19] を用いた KVS（netmap-kvs）[*2]，カーネル内で KVS 処理を行うために Netfilter を用いた KVS（netfilter-kvs）を比較対象とする．redis および netfilter-kvs はマルチスレッドに対応していない．また，実装した本システム以外の評価においても表 8 の環境を用いており，その場合，サーバ側の NIC は Intel X520-DA2 を用いている．

図 7 [*3] に実機による評価環境の外観図を示す．NetFPGA-10G の FPGA ボードを図中のホストマシンの PCI Express のスロットにマウントしており，ホストマシン側の NIC として動作させる．DAC（Direct Attached Cable）を用いて図中手前のクライアントマシンに接続されている．電力計測は，サーバマシンの AC アダプタにワットアワーメータ（SEW3A）[23] を挿入することで計測している．

6.2 スループット

クライアントマシン上で netmap フレームワーク [19] を用いてクエリインジェクションを行う．Key サイズを 64B，Value サイズを 64B とそれぞれ固定長でインジェクションを行う．実機評価で使用したインジェクションクラスは，**表 9** に示す 4 種類である．SET（HIT）のクエリタイプではすべて同一の Key を指定している．SET（MISS）では，ハードウェア上で必ず MISS となるようなロジックに

[*2] netmap は FreeBSD によるカーネルバイパスのフレームワークであるため，netmap-kvs での評価で使用した OS は FreeBSD R10.1-p29 である．
[*3] FPGA ボードへの電源供給および NIC への Reset 信号を伝送するためホスト PC にマウントして利用している．

表 8 評価環境
Table 8 Evaluation environment.

項目	クライアント	サーバ
CPU	Intel Core i5-4590	Intel Core i5-4460
Host Memory	8 GB	4 GB
OS	FreeBSD R10.1-p10	CentOS release 6.7
NIC	Intel X520-DA2	NetFPGA-10G

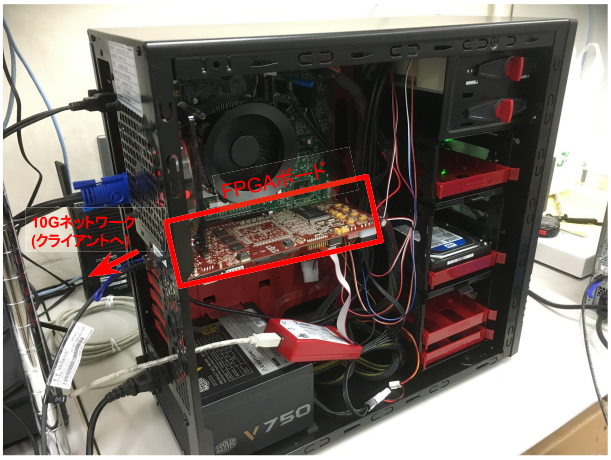

図 7 実機を用いた評価環境の外観図
Fig. 7 Appearance of evaluation environment using FPGA.

表 9 評価に使用したインジェクションクラス
Table 9 Injection classes for evaluations.

項目	備考
GET（HIT）	すべての GET リクエストが FPGA 内でヒットする
GET（MISS）	すべての GET リクエストが FPGA 内でミスする
SET（HIT）	すべての SET リクエストが FPGA 内でヒットする（既存の KeyValue を UPDATE）
SET（MISS）	すべての SET リクエストが FPGA 内でミスする（ソフト CPU 処理発生）

書き換えている．これはすべてクエリに対して，FPGA 内でメモリアロケーションが発生するようにしている．GET（HIT）ではすでに SET クエリで KeyValue のペアが登録されている状態を作り，すでに登録されている Key を指定して GET リクエストを送る．GET（MISS）では，あらかじめ SET クエリにより KeyValue を登録しない状態で計測を行う．したがって，すべての GET リクエストは MISS と判定され，Key に一致する Value がないことを示す応答パケットを返す．

表 10 に実機でのスループット評価の結果を示す．GET（HIT）では約 1.4 Mops となった．一方で GET（MISS）では GET（HIT）よりも 2 倍性能が良い．これは HashTable で Key が一致しないため，DataStore にアクセスする時間が含まれていないからである．SET（HIT）では 2.2 Mops の性能であった．一方で，SET（MISS）では，約 0.7 Mops

表 10 実機によるスループット評価 [単位：Mops]
Table 10 Throughput evaluation with FPGA [unit: Mops].

項目	平均スループット
GET（HIT）	1.42354
GET（MISS）	3.02158
SET（HIT）	2.20104
SET（MISS）	0.71049

であった．これは HashTable において新しいエントリの割当を行っているからである．そのメモリアロケーション処理を行っているソフト CPU の実行が Key-Value ペアの処理の流れを止めているために，スループットが低減している．

6.3 面積

本提案はデータ構造ごとに設計された PE をマルチコア化することでカスタム化および高速化を実現する．FPGA あたりに搭載できる PE 数を評価するために，PE 数を変化させたときの面積を評価する．対象とする FPGA ボードである NetFPGA-10G は Xilinx 社の Virtex-5 XC5VTX240T が搭載されている．総スライス数は 37k 個であり，それぞれのスライスは 6 入力 LUT とフリップフロップが 4 個ずつ含まれている．

本評価では，NetFPGA Project で提供されている Reference NIC の全体のスライスの割合および本設計の PE，クロスバスイッチ，DRAM コントローラを含むスライスの割合をそれぞれ論理合成を行い算出する．そして，その合計サイズをもとに複数の PE を論理合成した場合に，単一 FPGA にマッピングできる PE 数を評価する．論理合成の Optimize Mode は SPEED を選択した．

図 8 に論理合成による面積の結果を示す．Virtex-5 のスライス容量では PE を理論上 11 個まで実装できる[*4]．また，Reference NIC 単体で面積の約 48%を占めているが，実際の文字列型 PE 単体の面積は全体の約 3%程度であり，きわめて面積が小さいことが分かった．

6.4 消費電力

提案する PE の消費電力と関連技術の消費電力の評価を行う．今回はプロトタイプの実装の都合上，サーバマシンから FPGA ボードにリセットをかけているためサーバマシンが必要となった．そのため，計測した電力メータの値はホストマシンを含む全体の電力を含んでいるが，本論文ではデータ構造 PE がスタンドアロン動作を前提としている．それをふまえて，FPGA ボードに供給している電力を算出する．FPGA ボードをマウントせずにサーバマシン CPU のアイドル時の電力を P_{cpu} とする．同様に，FPGA

[*4] ただし，配置配線可能なスライス利用率は 100%よりも低いため，実際に実装できる PE 数はこれよりも少なくなる可能性がある．

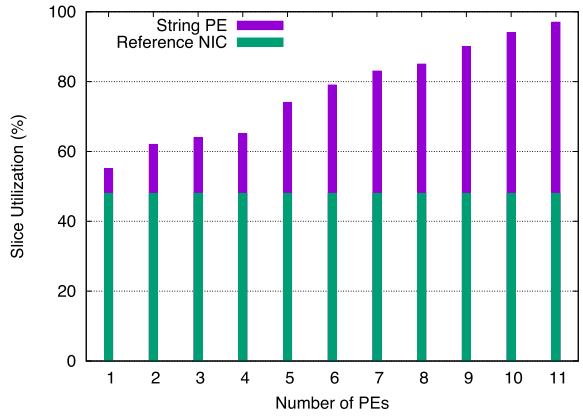

図 8 XC5VTX240T における PE 数ごとのスライス占有率
Fig. 8 Slice utilization on XC5VTX240T vs. number of PEs.

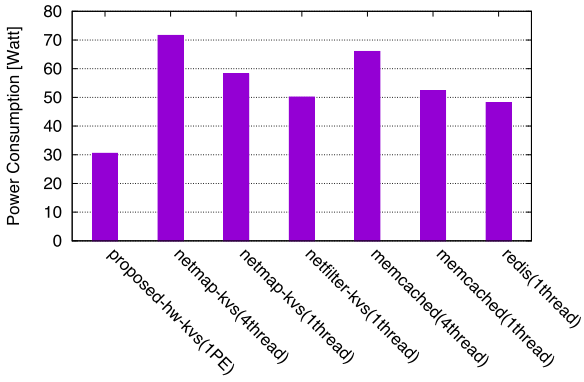

図 9 消費電力
Fig. 9 Power consumption.

ボードをサーバマシンにマウントした状態でスループット評価を行った状態での電力を $P_{cpu+fpga}$ とする．スタンドアロンで動作する FPGA の電力を P_{fpga} とすると，P_{fpga} は $P_{cpu+fpga} - P_{cpu}$ で求められる．その他のアプリケーションでは，P_{cpu} をその消費電力として扱う．

電力評価の結果を図 9 に示す．redis はシングルスレッド動作を想定したシステムであり 48.2 W であった．また memcached は CPU コア数分マルチスレッド化することで性能を向上させている．そのため，4 コア CPU のマシンでは消費電力が 66 W であった．netmap-kvs において CPU コアと同数である 4 スレッドで実行することで消費電力は 71.6 W となった．本提案ハードウェアによるシステムの消費電力はいずれのクエリタイプで負荷をかけた状態でも，約 30 W 程度であった．既存のデータベースシステムは CPU 上で動作しているため，専用ハードウェアと比較してその消費電力は大きい．本提案手法では FPGA 上に専用ハードウェアを実装しているため，ソフトウェアとして動作するデータベースシステムよりも低い消費電力で動作している．

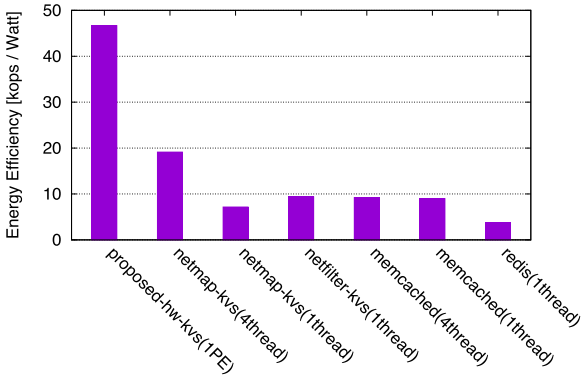

図 10 電力効率
Fig. 10 Energy efficiency.

6.5 電力効率

図 10 に提案手法および関連技術の電力性能比（kops/Watt）を示す．memcached はスレッド数に関係せず約 9.2 kops/W の結果であった．したがって，マルチスレッド化されていない redis ではマルチスレッド化による大幅な性能向上は見込めない．一方で，netmap-kvs は NIC から割込み要求を発行した CPU で処理を行っているため，4 スレッドで実行した際に電力効率は 1 スレッドに比べて 2.6 倍改善されている．同様に netfilter-kvs はマルチスレッド化することで，同等の性能改善が見込める．これらはすべて CPU で実行しているため，電力効率の観点から大きな改善が得られていない．一方で，専用ハードウェアによる本提案では 1PE の場合，2.4〜12.4 倍の電力効率の改善を達成している．

6.6 シミュレーション評価

6.2 節で行った 1PE による動作検証のスループット評価をもとに本研究で設計した PE 数を増加させて得られるスループットを試算する．コア数を増やすにあたり考慮すべき点として，DRAM コントローラのアービトレーションがあげられる．本システムはメモリコントローラを含めて 160 MHz で駆動する（ソフト CPU のみ 80 MHz で動作する）．Key 長が 64B，Value 長が 64B のとき，DRAM コントローラの実効性能を計算したところ，約 37 Mops であった．したがって，DRAM の律速値まで線形にスループットが増加する．これをもとにシミュレーション評価を行った．また，SET リクエストおよび GET リクエストにおける 10 Gbps ネットワークにおけるラインレート値を算出する．*Throughput* をラインレートとし，BW をネットワークの帯域とする．ここでは BW として 10 Gbps を仮定する．またネットワークのオーバヘッドのサイズを OH とする．OH は XGMII（10 Gigabit Media Independent Interface）におけるプリアンブル（8B），Ethernet ヘッダ（14B），IP ヘッダ（20B），UDP ヘッダ（8B），アプリケーションヘッダ（10B），Frame Check Sequuence（4B），Inter

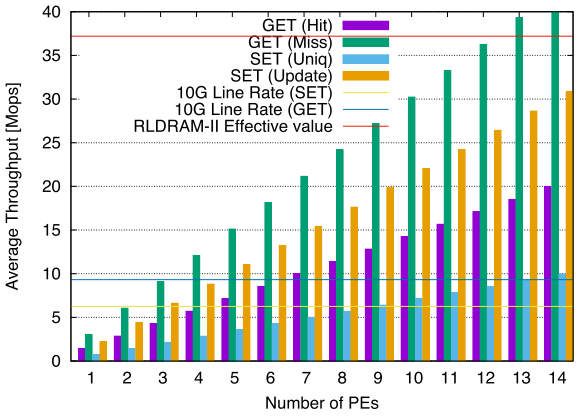

図 11 PE 数を増加させたときのスループット
Fig. 11 Throughput vs. number of PEs.

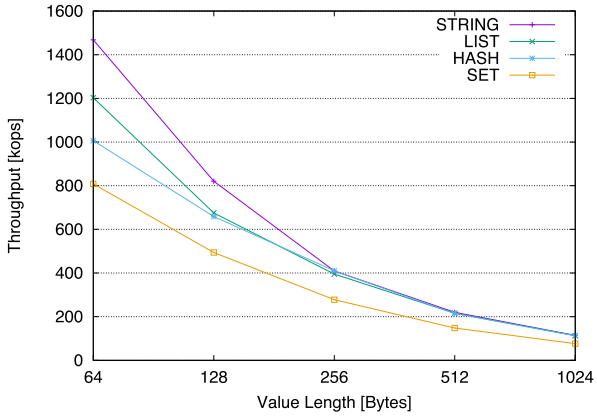

図 12 PE あたりの可変長 Value におけるスループット（READ 操作）
Fig. 12 Throughput per PE vs. value length (READ operation).

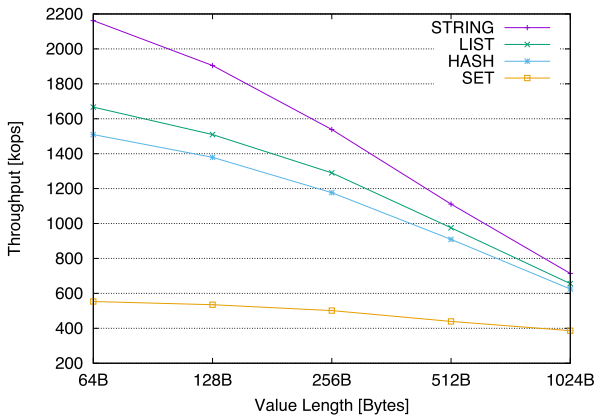

図 13 PE あたりの可変長 Value におけるスループット（WRITE 操作）
Fig. 13 Throughput per PE vs. value length (WRITE operation).

Frame Gap（12B）が含まれ，その合計値は 76B である．それぞれのフレームには *OH* に加えて *Payload* が含まれており，リクエスト時の Key や Value が含まれる．それを考慮に含めて以下にラインレートを算出する計算式を示す．

$$Throughput = BW/((OH + Payload) \times 8)$$

上の式より GET では Payload として Key である 64B が含まれ，スループットは 8.929 Mops となる．また，SET の Payload は Key の 64B および Value の 64B の合計値となり，スループットは 6.127 Mops となる．

図 11 に，評価結果を示す．それぞれのクエリタイプごとに PE 数を増やしたときの結果である．色で示す水平線は 10 Gbps ネットワークにおけるラインレートおよび DRAM の律速値を示している．10 Gbps ネットワークを想定した場合，GET リクエストを処理するには 7 コア，SET リクエストを処理するには 9 コアが必要であることを示している．

図 12 に各データ構造の 1PE あたりの READ クエリの Value サイズを変えたときのスループットを示す．**図 13** にはスラブアロケータ処理を含まない UPDATE 時[*5]の WRITE クエリのスループットを示す．

- LIST 型：LPUSH，LPOP といった操作は文字列型の操作に加えて，探索するアドレスの変更を HashTable に加える必要がある．そのため，HashTable のエントリの更新を行うメモリアクセスが発生し，文字列型よりもスループットが低くなった．
- HASH 型：Value を参照するために，Key と Field（Key とは異なる識別子）を用いている．Key と Field を直結したものをハッシュした値を HashTable 参照に用いている．そのため，HashTable のエントリを 2 つ用いており，メモリアクセスの回数が文字列型と LIST 型よりも多くなっている．
- SET 型：Key とペアとなる Value には member を追加することが可能であり，HashTable で管理されている．本設計では，member を管理する HashTable を chunk で管理している．member が追加されると，member の HashTable に member を格納している chunk のアドレスが追加される．したがって，Value（member）を参照するために Key の HashTable と member の HashTable のメモリアクセスが発生するため，他のデータ構造よりも低いスループットとなった．WRITE 操作においては，毎回アロケート処理を生じるため，ソフト CPU によるオーバヘッドを含むため，他のデータ構造よりも低いスループットとなった．

本論文では，深いパイプライン化による高性能化手法をとらず，今後拡張できるようにデータ構造ごとに PE を実装した．そのため図 12 および図 13 に示すようにデータ構造によって性能が異なるが，これらの性能をそれぞれマルチコア化することで高速化が可能である．6.3 節での面積評価で示したように本プロトタイプで用いた NetFPGA-10G

[*5] SET 型では Value を集合として扱うため，WRITE 時は必ずアロケート処理が生じる．

の FPGA では 11 コア程度まで搭載することができた．また，この FPGA ボードの次世代ボードである NetFPGA-SUME では Virtex-7 XC7V690T を搭載しており，より多くのコア数を搭載することが可能である．

6.7 議論

本論文で提案する KVS ではネットワーク I/O およびメモリ I/O が性能やデータ容量のスケーラビリティに影響を与える．これらを打破するための方策を以下に考察する．

- ネットワーク I/O：本プロトタイプで使用した FPGA ボードは 4 つの 10GbE のインタフェースを備えている．しかし，NetFPGA-10G では 11 コアまで搭載可能であるが，6.6 節で示したとおり 10 Gbps を処理するのに 9 コアが必要である．FPGA ボードの最大ネットワーク帯域である 40 Gbps に対応するには，FPGA を 10GbE MAC ごとに用意する方法が考えられる．なお，NetFPGA-10G の次世代ボードである NetFPGA-SUME には同容量のネットワーク帯域と Virtex-7 XC7V690T を搭載している．これは Virtex-5 のデバイスよりも多いため，より多くの PE をクロスバスイッチに接続でき，スループット性能をさらに改善できると考えられる．
- DRAM の容量：本プロトタイプでは，288 MB の RLDRAM-II を用いて KVS を構築した．FPGA ボード上にシステムを構築しているため，扱えるデータサイズが FPGA ボード上に搭載できる DRAM の容量によって制限されてしまう可能性がある．この場合，KVS を複数ノード（複数の FPGA ボード）で運用することで KVS の容量を増やすことも考えられる．また，5 章で紹介したようにアプリケーションによっては FPGA ボードに搭載されている DRAM の容量で足りる場合もあり，NetFPGA-SUME のようにさらに大容量のメモリを有する FPGA ボードを利用することでメモリ容量を拡大することもできる．

以上に示すように，今回はデバイスによる制約により，DRAM 容量は 288 MB に限定され，ネットワーク帯域においても 10GbE のネットワークに限定したシステムになった．しかし，最新の FPGA や複数の FPGA を用いることでこれらのスケーラビリティを改善できると考えられる．

7. まとめ

本論文では，従来の KVS の Value として多様なデータ構造に対応できるデータ構造サーバのハードウェア設計を提案した．新しいデータ構造への拡張を可能にするために各データ構造ごとに専用 PE を設計し，ヘテロジニアス・マルチコア方式を採用することで利用用途に応じて，各 PE の個数やデータ構造の種類をカスタマイズすることが可能となる．我々は，文字列型，LIST 型，HASH 型，SET 型の 4 種類の PE を設計し，プロトタイプ実装として FPGA 上に文字列型 PE を実装し動作確認を行った．動作確認では 4 種類のクエリパターンのトラフィックについて実機によるスループット評価を行った．その結果をもとに，PE 数を変えた場合のスループットの評価および面積評価を行った．スループット評価として 10 Gbps ネットワークのラインレートを処理するためには，7〜9 コアが必要であることが分かった．面積では Virtex-5 の FPGA において 11 コアまで搭載することが可能であった．また，消費電力は約 30 W であり，CPU 上で動作する KVS と比較して電力効率が最大 12.4 倍改善されることが分かった．以上の結果より，データ構造サーバであっても FPGA 上に実現することで十分なスループット性能を提供しつつ，センサデータ収集を対象としたネットワークアプライアンスに組み込み可能であるといえる．

謝辞 本研究の一部は，JST 戦略的創造推進事業さきがけ「多様な構造型ストレージ技術を統合可能な再構成可能データベース技術」の補助による．

参考文献

[1] Db-engines ranking, available from ⟨http://db-engines.com/en/ranking⟩ (accessed 2016-03-01).
[2] Blott, M., Karras, K.: Liu, L., Vissers, K., Baer, J. and Istvan, Z.: Achieving 10 Gbps Line-rate Key-value Stores with FPGAs, *Proc. USENIX Workshop on Hot Topics in Cloud Computing* (*HotCloud'13*) (June 2013).
[3] Blott, M., Liu, L., Karras, K. and Vissers, K.: Scaling Out to a Single-Node 80 Gbps Memcached Server with 40Terabytes of Memory, *Proc. USENIX Workshop on Hot Topics in Storage and File Systems* (*HotStorage'15*) (July 2015).
[4] Blott, M. and Vissers, K.: Dataflow Architectures for 10 Gbps Line-rate Key-value-Stores, *Proc. IEEE Symposium on High Performance Chips* (*HotChips'13)* (Aug. 2013).
[5] Chalamalasetti, S.R., Lim, K., Wright, M., Young, A.A., Ranganathan, P. and Margala, M.: An FPGA Memcached Appliance, *Proc. International Symposium on Field Programmable Gate Arrays* (*FPGA'13*), pp.245–254 (Feb. 2013).
[6] Cho, J.M. and Choi, K.: An FPGA Implementation of High-throughput Key-value Store Using Bloom Filter, *Proc. International Symposium on VLSI Design, Automation and Test* (*VLSI-DAT'14*), pp.1–4 (Apr. 2014).
[7] Danga Interactive, Memcached – A Distributed Memory Object Caching System, available from ⟨http://memcached.org/⟩ (accessed 2016-03-01).
[8] Fan, B., Andersen, D.G. and Kaminsky, M.: MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing, *Proc. USENIX Symposium on Networked Systems Design and Implementation* (*NSDI'13*), pp.371–384 (2013).
[9] Fukuda, E.S., Inoue, H., Takenaka, T., Kim, D., Sadahisa, T., Asai, T. and Motomura, M.: Caching Memcached at Reconfigurable Network Interface, *Proc. International Conference on Field-programmable Logic and Applications* (*FPL'14*), pp.1–6 (Sep. 2014).
[10] Hu, X., Wang, X., Li, Y., Zhou, L., Luo, Y.,

Ding, C., Jiang, S. and Wang, Z.: LAMA: Optimized Locality-aware Memory Allocation for Key-value Cache, *Proc. USENIX Annual Technical Conference* (*ATC'15*), pp.57–69 (July 2015).
[11] Istvan, Z., Alonso, G., Blott, M. and Vissers, K.: A Flexible Hash Table Design for 10 Gbps Key-value Stores on FPGAs, *Proc. International Conference on Field-programmable Logic and Applications* (*FPL'13*), pp.1–8 (Sep. 2013).
[12] Lavasani, M., Angepat, H. and Chiou, D.: An FPGA-based In-Line Accelerator for Memcached, Vol.13, No.2, pp.57–60 (2014).
[13] Li, S., Lim, H., Lee, V.W., Ahn, J.H., Kalia, A., Kaminsky, M., Andersen, D.G., Seongil, O., Lee, S. and Dubey, P.: Architecting to Achieve a Billion Requests Per Second Throughput on a Single Key-value Store Server Platform, *Proc. International Symposium on Computer Architecture* (*ISCA'15*), pp.476–488 (June 2015).
[14] Lim, H., Han, D., Andersen, D.G. and Kaminsky, M.: MICA: A Holistic Approach to Fast In-Memory Key-Value Storage, *Proc. USENIX Symposium on Networked Systems Design and Implementation* (*NSDI'14*), pp.429–444 (Apr. 2014).
[15] Lim, K., Meisner, D., Saidi, A.G., Ranganathan, P. and Wenisch, T.F.: Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached, *Proc. International Symposium on Computer Architecture* (*ISCA'13*), pp.36–47 (June 2013).
[16] Mitchell, C., Geng, Y. and Li, J.: Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store, *Proc. USENIX Annual Technical Conference* (*ATC'13*), pp.103–114 (June 2013).
[17] NetFPGA Project, available from ⟨http://netfpga.org/⟩ (accessed 2016-03-01).
[18] Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Parulkar, G., Rosenblum, M., Rumble, S.M., Stratmann, E. and Stutsman, R.: The Case for RAMClouds: Scalable High-performance Storage Entirely in DRAM, *ACM SIGOPS Operating System Review*, Vol.43, No.4, pp.92–105 (2010).
[19] Rizzo, L.: netmap: A Novel Framework for Fast Packet I/O, *Proc. USENIX Security Symposium* (*Security'12*), pp.101–112 (Aug. 2012).
[20] Salvatore Sanfilippo. Redis, available from ⟨http://redis.io/⟩ (accessed 2016-03-01).
[21] Xu, Y., Frachtenberg, E. and Jiang, S.: Building a High-performance Key-value Cache as an Energy-efficient Appliance, *Performance Evaluation*, Vol.79, pp.24–37 (Sep. 2014).
[22] 沖電気工業株式会社，入手先 ⟨https://www.oki.com/jp/rescat/⟩（参照 2016-03-01）.
[23] 株式会社システムアートウェア：ワットアワーメーター（shw3a），入手先 ⟨http://www.system-artware.co.jp/shw3a.html⟩（参照 2016-03-01）.

徳差 雄太

2014 年慶應義塾大学環境情報学部卒業．現在，同大学大学院理工学研究科修士課程在籍中．コンピュータアーキテクチャおよび FPGA システムに関する研究に従事．

松谷 宏紀 （正会員）

2004 年慶應義塾大学環境情報学部卒業．2008 年同大学大学院理工学研究科後期博士課程修了．博士（工学）．2009 年度より 2010 年度まで東京大学大学院情報理工学系研究科特別研究員，日本学術振興会特別研究員（SPD）．2011 年度より慶應義塾大学理工学部情報工学科専任講師．コンピュータアーキテクチャとインターコネクトに関する研究に従事．IEEE，電子情報通信学会各会員．