一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-83,CPSY2018-93,RECONF2018-57(2019-01)

FPGA NICを用いたEffectively-onceセマンティクスのための重複除去機構

鈴木　滉司†　三塚　皐矢††　岩田　拓真††　松谷　宏紀†,††

† 慶應義塾大学 理工学部 〒223-8522 神奈川県横浜市港北区日吉 3-14-1
†† 慶應義塾大学大学院 理工学研究科 〒223-8522 神奈川県横浜市港北区日吉 3-14-1
E-mail: †{suzuki,koya,iwata,matutani}@arc.ics.keio.ac.jp

あらまし　ストリーム処理やメッセージ配送システムにおける処理保証はat-most-once、at-least-once、exactly-onceに大別できる。送信側にat-least-onceによる再送機能を要求しつつ、受信側にて何らかの方法で処理の重複除去を行い、事実上のexactly-onceとするアプローチをeffectively-onceと呼ぶことがある。ストリームデータの増大や高効率化への要求から、ストリーム処理やメッセージキューをFPGA（Field Programmable Gate Array）ベースのNIC（Network Interface Card）上に実現する研究が多数報告されているが、そのようなシステムでは処理保証もFPGA NIC内で実現する必要がある。本論文ではFPGA NIC内に重複排除のためのハッシュテーブルを専用回路として実現し、一度処理したデータをハッシュテーブルに登録することで、同じデータが再送されたとしても、それを検出し、重複データを排除する。この方式であれば、冪等はない処理に対してもeffectively-onceをNICを通過する際に実現できる。FPGA NICを用いた実機評価の結果、無視できるほど小さい性能オーバヘッドでFPGA NIC内で重複除去できることを示した。

キーワード　ストリーム処理、重複除去、FPGA NIC

A Deduplication Mechanism for Effectively-once Semantics Using FPGA NIC

Koji SUZUKI†, Koya MITSUZUKA††, Takuma IWATA††, and Hiroki MATSUTANI†,††

† Faculty of Science and Technology, Keio University 3-14-1, Hiyoshi, Yokohama, JAPAN 223-8522
†† Graduate School of Science and Technology, Keio University 3-14-1, Hiyoshi, Yokohama, JAPAN 223-8522
E-mail: †{suzuki,koya,iwata,matutani}@arc.ics.keio.ac.jp

Key words　Stream processing, Deduplication, FPGA NIC

1. はじめに

ストリーム処理は、永続的に発生する時系列データ（ストリームデータ）に対し、単一もしくは複数の演算を繰り返す処理方法である。また、メッセージキューは、送信側（プロデューサ）と受信側（コンシューマ）の間のメッセージ配送を司る非同期キューである。近年のセンシング技術、IoT（Internet of Things）技術、SNS（Social Networking Service）の進捗と普及によって、メッセージキューのように大量のストリームデータの収集および配送を行う技術、および、ストリーム処理のようなデータ処理技術が広く利用されている。ストリーム処理のフレームワークとしてApache Storm [1]、Apache Spark Streaming [2] などがあり、メッセージキューとしてはApache Kafka [3] などが有名である。実用的なストリーム処理やメッセージ配送システムにおいては、処理保証のレベルが事前に定められている。一般的に、処理保証にはat-most-once、at-least-once、exactly-onceがある。at-most-onceでは「処理」は必ず1回以下実行されるが、at-least-onceでは必ず1回以上実行される。exactly-onceでは必ず1回実行されるがスループットが犠牲になることが多い。またストリーム処理には、1) ストリーム中のデータ1つ1つに対して処理を行うone-at-a-time方式、および、2) ストリームデータを短い時間間隔で区切ったmicro-batchごとに処理を行うmicro-batch方式がある [4]。one-at-a-time方式のストリーム処理フレームワークの多くはat-least-once処理となる。一方、micro-batch方式ではexactly-onceをサポートするものもあるが、バッチ処理に強い順序付けが必要となるケース

もある [4]。

低コストで exactly-once を実現する方法として、at-least-once 方式と冪等な処理を組み合わせることで、事実上の exactly-once とすることもできる。このように送信側に at-least-once 方式を要求しつつ、受信側にて何らかの方法で処理の重複除去を行い、事実上の exactly-once とするアプローチを effectively-once と呼ぶこともある。本研究ではこの effectively-once を効率的に実現することに焦点を当てる。

近年のストリーム処理技術の進展に加え、ストリームデータの増大にしたがい、ストリーム処理やメッセージキューを少ない計算リソース、および、少ない消費電力で実現することの重要性が増している。このために、FPGA（Field Programmable Gate Array）ベースの NIC（Network Interface Card）を用いて、ストリーム処理を行う研究 [5]、メッセージキューを行う研究 [6]、シリアライゼーションを行う研究 [7] などが報告されている。このようにストリーム処理やメッセージ配送が NIC で実現されるようになると、処理保証も NIC 内で実現する必要があるが、これらの既存研究では処理保証は考慮されていない。

本研究では、NIC 内ストリーム処理や NIC 内メッセージ配送での利用を想定した effectively-once 手法を提案する。具体的には、1) 入力データごとに一意な ID が付与されていること、および、2) 送信側が at-least-once を前提に再送をすること、この 2 点を前提として、FPGA ベースの 10Gbit Ethernet（10GbE）NIC 上に重複排除のためのハッシュテーブルを専用回路として実現する。一度処理したデータはこのハッシュテーブルに記憶されるため、同じデータが再送されたとしても、それを検出し、重複データを排除できる。この方式であれば、冪等ではない処理に対しても effectively-once を NIC 内で自動的に実現できる。

本論文の構成は以下の通りである。2. 章では、本論文の関連技術を述べる。3. 章で本論文の提案内容について述べる。4. 章で本論文が提案する FPGA NIC を用いた重複除去機構の実装について述べる。5. 章で評価を述べ、6. 章で本論文のまとめと今後の課題を述べる。

2. 関 連 技 術

2.1 ストリーム処理と処理保証

様々なストリーム処理フレームワークが登場しており、それらの特徴もまた様々である。例えば、Apache が提供しているストリーム処理フレームワークについては [8] が詳しい。

前述のとおり、処理保証は at-most-once、at-least-once、exactly-once の 3 種類に分類でき、多くのストリーム処理フレームワークは at-least-once もしくは exactly-once を採用している。全体的な傾向として、one-at-a-time 方式のストリーム処理フレームワークは at-least-once を前提とすることが多く、micro-batch 方式のストリーム処理フレームワークにおいては exactly-once をサポートするものがある。例えば、one-at-a-time 方式のストリーム処理フレームワークである Apache Storm [1] は at-least-once を前提とするが、それを Trident API を用いて micro-batch 化したものは exactly-once をサポートする。micro-batch 方式の Apache Spark Streaming [2] も exactly-once をサポートする。

exactly-once は「正確に一回送信し受信する」ことを保証するものだが、最終的に同様の結果を得るための実現方法にはいくつかのアプローチがある。例えば、

- データの送信側は必要に応じてデータを再送可能としたうえで、
- 受信データを、まず、write-ahead log としてチェックポイントを管理する永続ストレージに書き込んだ後に処理することで受信側にとっての exactly-once を実現することもできる。当然、ストリーム処理の際に一定のオーバヘッドが生じることになる。

2.2 メッセージキューと配送保証

メッセージ配送システムにおいては、プロデューサのクラッシュ、ネットワークの切断、ブローカのクラッシュなどによってメッセージが正しく配送できない可能性があることから、配送保証のレベルを定めておくことは重要である。配送保証のレベルは at-most-once、at-least-once、exactly-once に分類できる。例えば、Apacke Kafka [3] は当初から at-least-once をサポートしていたが、近年ではプロデューサとブローカ（Kafka）間、および、ブローカ（Kafka）とコンシューマ間の状態を管理する機能を導入することで、exactly-once をサポートできるようになっている。

同じくメッセージ配送システムである Apache Pulsar [9] では、メッセージの重複除去によって事実上の exactly-once を実現している。具体的には、

- メッセージ毎に固有のメッセージ ID が付加されているものとし、
- ブローカ（Pulsar）は各プロデューサが最後に正常に送信できたメッセージ ID を追跡することで、重複排除を行う。

図 1 に Apache Pulsar におけるメッセージの流れを示す。Apache Pulsar では永続ストレージの使用によってメッセージ ID の追跡を可能にしている。具体的には、永続ストレージとして Apache BookKeeper [10] を用いてメッセージデータを保存し、コンシューマに配送されて受信確認を受け取るまでデータを保持し続ける。コンシューマがどこまで購読したかを示すカーソルも永続ストレージに保持される。これらの情報を永続ストレージに保存することで、ブローカがクラッシュした後に復旧したとしてもこれらの情報を復元できるようになっている。従来のシステムではブローカはプロデューサがクラッシュして再接続してきた際に新規のユーザと見做すが、Pulsar の場合は各プロデューサを認識している。また、連続性を維持しているため、プロデューサがクラッシュした場合もそれぞれが最後に正常に送信できたメッセージ ID を知ることができ、前回の続きから再開することが可能である。

Apache Pulsar のように「メッセージ配送システムが重複したメッセージを検出および破棄し、100%の精度で処理できること」を強調した方式を、exactly-once を念頭に、effectively-once と呼ぶことがある。この場合、プロデューサはメッセージを重複して送信してもよく（at-least-once）、ブローカ側（Pulsar）

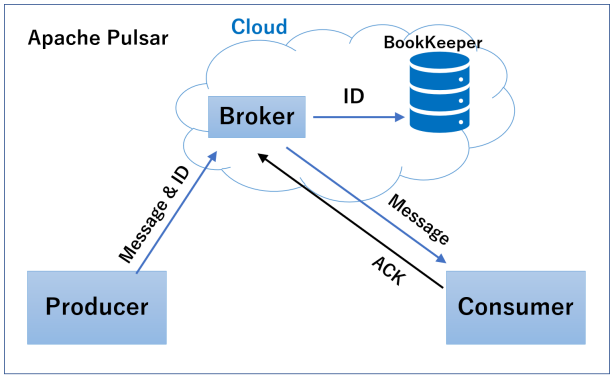

図 1　Apache Pulsar におけるメッセージの流れ

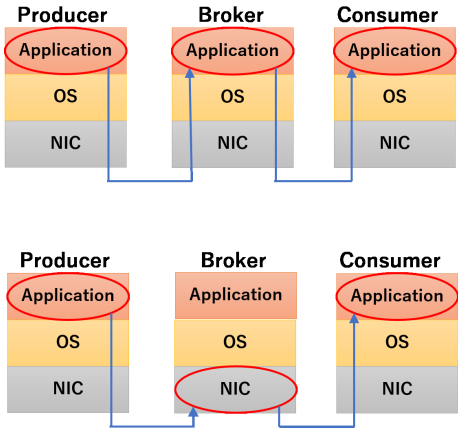

図 2　メッセージ配送システムにおけるメッセージの流れ（上図がアプリケーション層での処理、下図が NIC での処理 [6]）

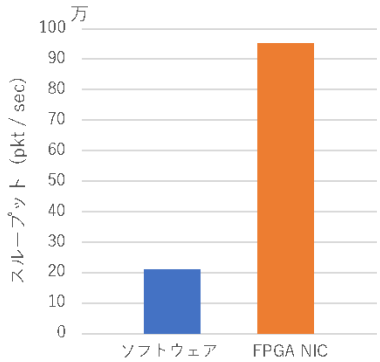

図 3　アプリケーション層および FPGA NIC によるストリーム処理性能

が重複排除を行うことにより過不足なくメッセージを配送もしくは処理する。本論文でもこの方式を effectively-once と呼ぶ。

2.3　ソフトウェアによる Effectively-once の実現

上述の effectively-once を実現するためのアプローチはいくつか考えられる。データの送信側は必要に応じてデータを再送可能であることを前提としたとき、このためのアプローチ例を以下に 3 つ紹介する。

（1） 冪等な処理に限定する。

（2） 受信側にて、正常に処理できたメッセージの情報を永続ストレージに保持することで追跡可能とし、重複除去を行う。

（3） 受信側にて、正常に処理できたメッセージの情報を Key-Value Store （KVS）に保持することで追跡可能とし、重複除去を行う。

(1) であれば重複除去すら不要である。ただし、対象とする処理は冪等でなければならず、カウンタのインクリメントなど単純にリトライしてしまうと結果が不正確になってしまう処理については注意が必要である。

(2) は Apache Pulsar が採用している手法に近い。永続ストレージの利用によって、重複除去の耐故障性を高めている。

(3) は (2) の簡易実装である。例えば、メッセージ ID を Key とし、メッセージを処理済みか否かの情報を Value とすることで、正常に処理できたメッセージを追跡可能とする。このような KVS をメモリ上に実現することでメッセージの重複を高速に判定できるが、耐故障性が犠牲になる。

3.　設　　計

3.1　提案の背景

2.3 節で述べた重複除去は通常、ソフトウェアとしてアプリケーション層で実現される。これは、通常のストリーム処理フレームワークやメッセージ配送システムは、ソフトウェアとしてアプリケーション層に実現されるためである。メッセージ配送システムを例に、メッセージの流れを図 2 に示す。上図が通常のメッセージの流れである。

一方で、ストリームデータの増大にしたがい、ストリーム処理やメッセージキューを少ない計算リソース、および、少ない消費電力で実現することの重要性が増しており、FPGA ベースの NIC（FPGA NIC）を用いて、ストリーム処理を行う研究 [5]、メッセージキューを行う研究 [6]、シリアライゼーションを行う研究 [7] などが報告されている。例えば、文献 [6] ではメッセージキューのキャッシュを FPGA NIC 上に実現しており、出版レートと購読レートの均衡が取れているときは図 2 の下図のような動作となる。また、文献 [5] ではワードカウントなどのアプリケーションを FPGA NIC 上で実現している。

一例として、本論文でも類似のワードカウントアプリケーションを FPGA NIC 上に実現し、通常通りアプリケーション層で実行した場合と処理スループットを比較した。実験には 100 万個のパケットを使用し、各パケットには 3 文字のワードが格納されているものとする。実装および評価環境はそれぞれ 4. 章と 5. 章で説明するとして、ここでは処理スループットの違いを図 3 に示す。ストリーム処理をアプリケーション層で行う場合と NIC で行う場合の処理スループットに明確な差が存在することが分かる。

ストリーム処理やメッセージ配送が NIC で実現されるようになると、処理保証も NIC 内で実現する必要があるが、文献 [5]、[6]、[7] などの FPGA NIC を前提とした既存研究では処理保証は考慮されていない。また、ワードカウントは冪等にしにくい処理であり、これで effectively-once を実現しようとすると、送信側による再送に加え、受信側による重複除去が必要

になる。つまり、FPGA NIC 上での重複除去の仕組みが必要である。このためのアプローチとして、例えば 2.3 節で述べた (1) から (3) の方法があるが、(1) では冪等ではない処理に対応できないので対象外とした。(2) と (3) を比べると、(2) のほうが耐故障性が高いが、FPGA NIC 内に専用回路として実現することを考えると、より単純な (3) のほうが有利である。実際、FPGA NIC を対象とした高効率な KVS 処理は多数研究されている [11] [12]。

3.2 提 案 手 法

本研究では FPGA NIC 上に effectively-once を実現することを提案する。具体的には、1) 入力データごとに一意な ID が付与されていること、および、2) 送信側が at-least-once を前提に再送をすること、この 2 点を前提として、10GbE FPGA NIC 上に重複排除のためのハッシュテーブルを専用回路として実現する。一度処理したデータはこのハッシュテーブルに記憶されるため、同じデータが再送されたとしても、それを検出し、重複データを排除できる。

3.3 ハッシュテーブルを用いた重複除去

本節では、ハッシュテーブルを用いた重複除去機構を実装していく上で必要な処理、及び考慮しなければならない項目について述べる。まず本研究では入力データごとに一意な ID が付与されているが、重複除去機構にはこの ID が入力として与えられる。ID から算出されたハッシュ値がハッシュテーブルのアドレスとなり、元の ID がハッシュテーブルに記憶され重複除去に用いられる（$Key = Hash(ID), Value = ID$）。ハッシュ値が衝突する可能性があるが、ID は入力データごとに一意であるため、実装すべき処理は「同じ内容のデータ」を受信した場合と「同じハッシュ値をもつ違う内容のデータ」を受信した場合に大きく分類して考えることができる。「同じ内容のデータ」の場合、重複した入力であると判断し、テーブルへの書き込みは行わず受信パケットも破棄する。パケット単位でのデータ破棄の方法については 4. 章にて触れる。対して「同じハッシュ値をもつ違う内容のデータ」の場合、適切な制御が必要となる。データの重複は連続して起こると想定されるため、衝突が発生する前に重複の除去は完了していると考えられる。本論文ではこの想定に基づき単純に上書きするものとして実装評価した。

以上の内容を踏まえると、パケットは以下の順に処理される。

（1） メッセージの情報（本研究における ID）よりハッシュ値を算出する。

（2） 算出したハッシュ値と同じ値を Key としてもつハッシュテーブルの中身を参照する。

（3） 受信パケットの ID と読み出した ID を比較する。

（4） 重複した入力と判断した場合、該当パケットを破棄する。新規の入力と判断した場合、ハッシュテーブルを更新して該当パケットを受理する。

本論文ではこの設計方針に基づいて実装・評価を行っている。

4. 実　　装

4.1 重複除去機構の FPGA 実装

本節では、重複除去機構を NIC として実装する方法について述べる。本論文では 10GbE インターフェースを持つ FPGA ボードとして NetFPGA-SUME を採用した。 NetFPGA-SUME の主な仕様を表 1 に示す。10GbE MAC は Xilinx 社によって提供される IP コアを、その他スイッチとしての機能全般は、NetFPGA チーム [13] によって提供されている設計データを用いて FPGA 上に構成した。本論文の実装は、この NetFPGA チームによって提供されている Reference NIC の HDL コードの一部に組み込んだものである。

表 1　NetFPGA-SUME の主な仕様

FPGA	Xilinx Virtex-7 TX690T
10GbE インターフェース	4 SFP+ connectors (10Gb × 4)
動作周波数	160MHz

実装した NIC のブロック図を図 5 に示す。NetFPGA-SUME の AXI（Advanced eXtensible Interface）の実装に従って、パケットの入力及び出力は 256bit ずつ行われ、それよりも大きいパケットは分割されて処理される。受信パケットの様子を図 4 に示す。パケットは先頭から順に Ethernet ヘッダ、IP ヘッダ、UDP ヘッダ、ID、データ本体と並んでいる。まず 4 つの 10GbE ポートとホストマシンから入力されるパケットに対して Input Arbiter が調停を行う。次に Outputport Lookup モジュールにてパケットの情報から送信先のポートを決定する。その後データを本論文で実装したモジュールへ送り、3.3 節で示したようにハッシュ値からテーブルのデータを取得し、入力パケットがもつ ID との比較を行う。

（1） 一致した場合は重複した入力と判断し、データを破棄する。

（2） 不一致ならば新規の入力と判断し、テーブルを上書きしてデータを送信する。

破棄されなかったデータは Output Queue に入れられ、適切なポートへと送信される。本研究では入力が重複しているか否かは ID で判断しているが、NetFPGA-SUME においては図 4 に示すように ID が含まれているのは 2 フリット目となる。そのため、パケット全体を重複排除するためには 1 フリット目を送信する前に 2 フリット目の中身を参照する必要がある。よってモジュールとして実装する際に性能を落とさないためにはパイプライン化が必須となる。正しい重複除去を行うために、パイプラインの進行には次のフリットの到着を条件として加える必要がある。

続いて実装モジュールの概要図を図 6 に示す。本論文ではハッシュテーブルは簡易的に Block RAM として実装している。テーブルのアドレス幅は 18bit、データ幅は 32bit とした。入力はまず全て FIFO キュー (図中の Input queue) に収められた後、重複除去機構に送られる。ここで実装したモジュールで処理が必要かどうかを判定する。入力データは以下の順に処理される。

（1） 先頭の 256bit が Input queue の出力として到着する。図 4 に示す通り、このフリットには重複除去に必要な情報 (ID) をもたないため特別な処理は必要ない。

（2） 次の 256bit が Input queue の出力として到着する。こ

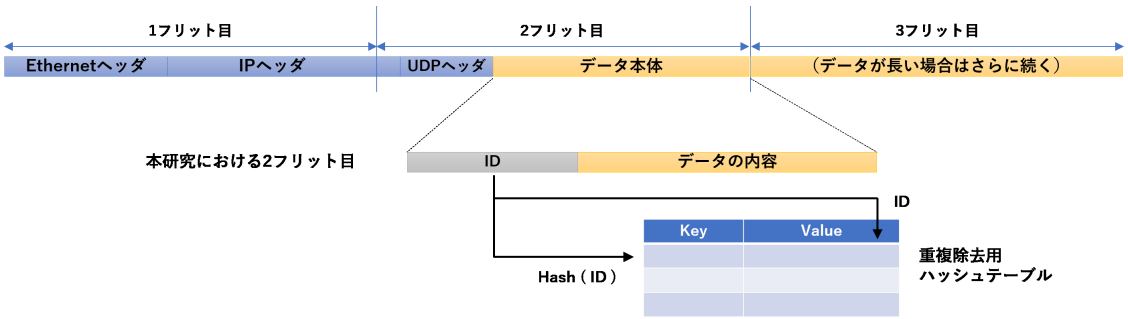

図 4　受信パケットの扱い

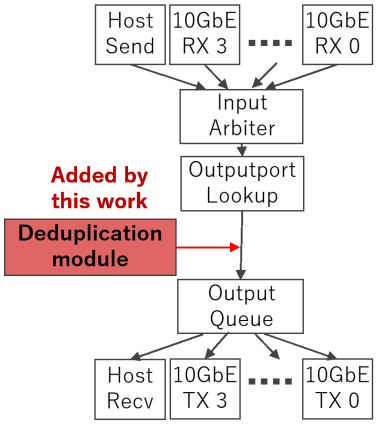

図 5　FPGA NIC のブロック図

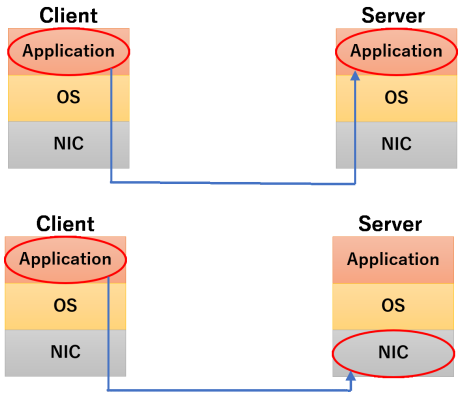

図 7　ストリーム処理におけるメッセージの流れ

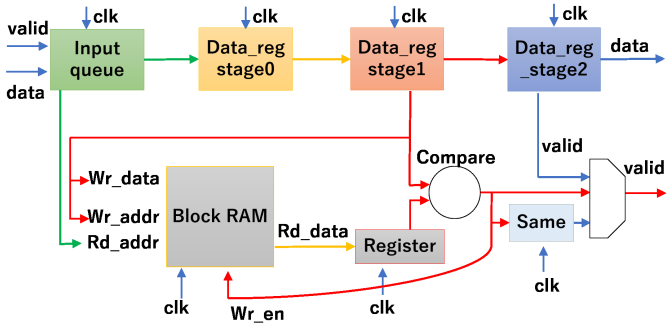

図 6　実装モジュールの概要図

のフリットは ID を含むためハッシュ値の算出に用いる。ハッシュ値はテーブル（Block RAM）の読み出しアドレスとなる。

（3） テーブルから読み出したデータが到着する。このデータは次のステージまで保持する。

（4） 入力データと読み出したデータを比較し、データの送信やテーブルへの書き込みの可否を決定する。

これらの処理はそれぞれを 1 サイクルとしたパイプラインで実行される。

先頭の 256bit は（4）の時点で図 6 中の Data_reg_stage2 レジスタに位置し、次のクロック立ち上がりで次のモジュールへと送信される。NetFPGA-SUME における各モジュールの通信はハンドシェイク方式を用いているため、ハンドシェイク用信号（図中の出力信号 valid）の算出に比較の結果を用いることでパケット単位でのデータの破棄が間に合う。NetFPGA-SUME では各パケットの最終フリットか否かを示す入力信号が存在しているため、比較の結果をレジスタ（図中の Same）にて最終フリットまで保持することで過不足なくデータを破棄することが可能である。

5. 評　　価

5.1 評 価 環 境

評価環境を表 2 に示す。本論文では 2 台の PC 間（MachineB → MachineA）でソケット通信を行う C 言語プログラムを用いて評価を行った。

表 2　実験に使用したマシンの詳細

	CPU	OS	NIC
MachineA	Intel Core i5-4460 (3.20GHz, 4cores)	Ubuntu 16.04 LTS	NetFPGA-SUME
MachineB	Intel Core i5-7500 (3.40GHz, 4cores)	Ubuntu 18.04.1 LTS	Mellanox ConnectX-3 Pro EN 10GbE

実験を行ったストリーム処理の概要を図 7 に示す。アプリケーションがソフトウェアとして動作している場合は重複除去もソフトウェアで実現可能である。3.1 節で述べたように高性能化のためにアプリケーションを NIC にオフロードすることもできるが、この場合重複除去も NIC にて行う必要がある。送信側のソフトウェアは 32bit の ID と 3 文字 (24bit) のワードを含むパケットを送信し、受信側は 100 万個のパケットを受信・処理するのにかかった時間を計測する。この評価における重複とは ID の重複のことを指す。続いて冪等ではない処理を

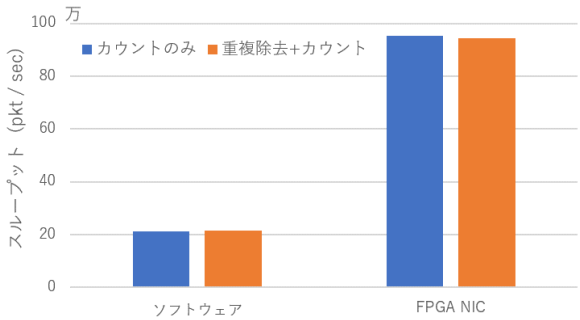

図 8　ワードカウントの処理スループット

行うアプリケーションとして、ワードカウントをそれぞれアプリケーション層ソフトウェアと FPGA NIC に実装した。動作は大まかに以下のとおりである。

（1） 必要があれば ID を用いた重複除去を直前のモジュールまたはソフトウェアで行う。

（2） 受け取ったパケットのワードを照合し、各ワードの処理回数を+1 する。

（3） パケットを 100 万個受信したら通信を終了する。終了時に処理にかかった時間を出力する。

FPGA NIC で処理される場合もソフトウェアで時間計測を行っているため、その分のオーバーヘッドが含まれた値が観測されている。

5.2　提案手法のスループット

FPGA NIC とソフトウェアでそれぞれ重複除去をした場合、しなかった場合のワードカウントアプリケーションのスループットを図 8 に示す。なお対象実機である NetFPGA-SUME の動作周波数は表 1 に示す 160MHz である。本論文ではスループットをアプリケーション (NIC またはソフトウェア) が 1 秒間に処理できたパケット数と定義している。結果、FPGA NIC での実装とソフトウェアでの実装の間には約 4.5 倍の性能差が存在していることがわかる。

また、本研究では重複除去機構を NIC に実装したが、既存のアプリケーションに組み込んだ際に重複除去機構自身がボトルネックとなり性能向上を阻害する可能性がある。そのため、NIC 上で動作するアプリケーションの、重複除去機構の有無によるスループットの差を確認する必要があるが、図中右側で示すように両者のスループットに大きな差は見られなかった。

6.　まとめと今後の課題

本論文では、FPGA NIC 内でのストリーム処理やメッセージ配送での利用を想定した effectively-once 処理保証の手法として、NIC 上にハッシュテーブルによる重複排除機構を専用回路として実装した。これにより、NIC がもつ高いスループットをほぼ損なうことなく冪等性のないアプリケーションを冪等な処理として扱うことを実現した。

本研究では「同じハッシュ値をもつ違う内容のデータ」を単純に新規の入力であると判断した。この方法では、ある入力 A に対して重複の除去を保証する期間が「同じハッシュ値をもつ別の ID の入力 B を受信するまで」という予測できない期間となるため、ストリームの内容に依存している部分がある。基本的に重複したデータは続いて到着することが多いとはいえ、同じハッシュ値を持つデータが A、B、A の順にやってきた場合に重複として排除すべき 2 回目の入力 A は比較対象が 1 回目の A ではなく B となるため新規の入力として扱われてしまう。

衝突率を軽減する手段として、例えば DRAM のような大容量のメモリが利用できる。また異なるハッシュ関数を用いたハッシュテーブルによる重複除去を複数回に渡って実行していくことで一つのテーブルにおけるハッシュの衝突の影響を軽減でき、精度向上につながる。

さらなる対策として、ハッシュテーブルへ書き込む内容にタイムスタンプを加えることなどが挙げられる。あるアドレス（ハッシュ値）を用いてテーブルへの上書きを行う際、前回の書き込みから規定の時間が経過しているかどうかを条件に加えることで 1 つのデータの重複排除について確実な保証期間を設けることができる。ただしこの場合、規定の時間が経過していなかった場合の重複データの処遇についても考慮する必要がある。

これらの工夫を取り入れた実装を行うことで、より確実な保証のある通信を実現できる。

文　献

[1] “The Apache Storm”. http://storm.apache.org/.
[2] “The Apache Spark Streaming”. http://spark.apache.org/streaming/.
[3] “The Apache Kafka”. http://kafka.apache.org/.
[4] N. Marz and J. Warren, Big Data: Principles and Best Practices of Scalable Real-Time Data Systems, Manning Publications, March 2016.
[5] K. Nakamura, A. Hayashi, and H. Matsutani, “An FPGA-Based Low-Latency Network Processing for Spark Streaming,” Proceedings of the 4th IEEE International Conference on Big Data (BigData’16) Workshops, pp.2410–2415, Dec. 2016.
[6] K. Mitsuzuka, Y. Tokusashi, and H. Matsutani, “Multi-MQC: A Multilevel Message Queuing Cache Combining In-NIC and In-Kernel Memories,” Proceedings of the 17th IEEE International Conference on Field Programmable Technology (ICFPT’18), Dec. 2018.
[7] T. Iwata, K. Nakamura, Y. Tokusashi, and H. Matsutani, “Accelerating Online Change-Point Detection Algorithm using 10GbE FPGA NIC,” Proceedings of the 24th International European Conference on Parallel and Distributed Computing (Euro-Par’18) Workshops, Aug. 2018.
[8] I. Hellstrom, “An Overview of Apache Streaming Technologies”. https://databaseline.bitbucket.io/an-overview-of-apache-streaming-technologies/.
[9] “The Apache Pulsar”. http://pulsar.apache.org/.
[10] “The Apache BookKeeper”. http://bookkeeper.apache.org/.
[11] Y. Tokusashi and H. Matsutani, “A Multilevel NOSQL Cache Design Combining In-NIC and In-Kernel Caches,” Proceedings of the International Symposium on High Performance Interconnects (Hot Interconnects 24), pp.60–67, Aug. 2016.
[12] M. Blott, K. Karras, L. Liu, K. Vissers, J. Baer, and Z. Istvan, “Achieving 10Gbps Line-rate Key-value Stores with FPGAs,” Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud’13), June 2013.
[13] “The NetFPGA Project”. http://netfpga.org/.