Apache Drill と FPGA アクセラレータを統合するシステムアーキテクチャの提案

渡辺 聡 [1]　仲川 和志 [2]　藤川 義文 [3]
在塚 俊之 [4]　小崎 信明 [5]　藤本 和久 [6]

Field Programmable Gate Array（FPGA）を用いたデータベース（DB）のアクセラレータシステムは，アクセラレータを適用しない場合と比較して，処理性能と消費電力を大幅に向上する可能性を有している．しかしながら，DB のアクセラレータシステムの実用化に向けては各種の課題が存在し，先行研究において，探求すべきオープンプロブレムが示されている．オープンプロブレムの解決に向けて，本研究では，オープンソースの DB である Apache Drill（Drill）と FPGA アクセラレータを統合するシステムアーキテクチャを提案する．提案するシステムアーキテクチャは，Drill のソースコード改変が不要，Drill と FPGA アクセラレータの高速なデータ通信，という二つの特徴を有する．また，本稿では，提案するシステムアーキテクチャによるオープンプロブレムの解決方法について論じる．

1　はじめに

データ分析処理の高速化を目的として，カラムストアデータベース（DB）が広く用いられている．カラムストア DB は，データの圧縮率向上，および，データの部分読み出しにより，高速処理を実現する [1, 15]．また，カラムストア DB の性能向上手段として，インメモリシステムが用いられている．インメモリシステムは，全てのデータを Dynamic Random Access Memory（DRAM）に格納することにより，ストレージからデータを抽出するオーバヘッドを削減する．

DRAM は優れた応答性能を有する一方，ビット単価，および，消費電力の面では Solid State Drive（SSD）に劣る．SSD に対する DRAM のビット単価は約 10 倍である [4]．消費電力の面では，4GB の DRAM は 40W を消費し，64GB の SSD は 0.2W から 10W を消費する（SSD は待機時間に消費電力を抑制する）[18]．従って，SSD に対する DRAM の消費電力量は 4 倍から 200 倍である．同一容量で比較した場合の SSD に対する DRAM の消費電力量は，64 倍から 3200 倍である．

ビット単価・消費電力の削減と，高速化の両立のためには，SSD からのデータ抽出の高速化が求められる．この実現手段として，我々は Field Programmable Gate Array（FPGA）を用いたアクセラレータを提案している [20]．FPGA はデバイスの製造後にハードウェア（HW）回路を設定可能なデバイスである．FPGA には処理内容に特化した HW 回路を設定可能であり，汎用の HW 回路である CPU と比較して，性能向上と消費電力削減が可能である [8, 9]．

DB のアクセラレータシステムの実用化に向けては各種の課題があり，探求すべきオープンプロブレムが示されている [5]．オープンプロブレムの解決に向けて，本研究では，オープンソースの DB である Apache Drill（Drill）と FPGA アクセラレータを統合するシステムアーキテクチャを提案する．本稿では，2 章から 4 章において，提案するシステムアーキテクチャを説明し，5 章において，オープンプロブレムの解決方法について論じる．

本研究で提案するシステムアーキテクチャは，(1)Drill のソースコード改変が不要，(2)Drill と FPGA アクセラレータの高速なデータ通信，という二つの特徴を有する．(1) に関しては，ソースコード改変による開発・導入コストの増大を抑制するために重要な観点である．先行研究において，GROUP BY 句による集約処理を FPGA にオフロードすることで高い性能向上効果が得られることが示されている [21]．しかしながら，従来研究では，DB のソースコードを改変して，集約処理のオフロードを実現していた [20, 21]．(2) に関しては，FPGA アクセラレータの高速性を発揮するために必要不可欠な観点である．SSD から FPGA アクセラレータに対する高速なデータ供給方法に関しては [20] において説明した．本稿では，FPGA アクセラレータの処理結果を Drill に対して高速に供給する方法を示す．

本研究の主な貢献は以下の 3 点である．

- DB のアクセラレータシステムの実用化を推進するため，オープンプロブレムに対する解決策となるシステムアーキテクチャを提示した点
- 提案するシステムアーキテクチャにおいて，アクセラレータの適用効果の高い集約処理を Drill のソースコード改変なしに実現した点
- FPGA アクセラレータの処理結果を高速に Drill に供給するための方法を示し，FPGA アクセラレータによる Drill の高性能化を可能にした点

2　準備

2.1　FPGA アクセラレータ

本稿のシステムアーキテクチャの提案は，我々が提案している FPGA アクセラレータを想定して行う．提案する FPGA アクセラレータは以下の三つの特性を有する．

(i)　ストレージからのデータ抽出を高速化する
(ii)　FPGA アクセラレータはステートレスである
(iii)　FPGA アクセラレータとサーバ装置は DMA を用いて通信

[1] 正会員　株式会社日立製作所 デジタルテクノロジーイノベーションセンタ
satoru.watanabe.aw@hitachi.com
[2] 非会員　株式会社日立製作所 デジタルテクノロジーイノベーションセンタ
kazushi.nakagawa.pr@hitachi.com
[3] 非会員　株式会社日立製作所 デジタルテクノロジーイノベーションセンタ
yoshifumi.fujikawa.su@hitachi.com
[4] 非会員　株式会社日立製作所 デジタルテクノロジーイノベーションセンタ
toshiyuki.aritsuka.kw@hitachi.com
[5] 非会員　株式会社日立製作所 デジタルテクノロジーイノベーションセンタ
nobuaki.ozaki.qr@hitachi.com
[6] 非会員　株式会社日立製作所 デジタルテクノロジーイノベーションセンタ
kazuhisa.fujimoto.bw@hitachi.com

可能である

(i) におけるデータ抽出とは，Filter 処理と Aggregation 処理の二つの処理である．Filter 処理は，SQL 文の実行に必要なデータをストレージから読み出し，WHERE 句と SELECT 句に従って，行やカラムを選択する処理である．また，Aggregation 処理は，Filter 処理で選択したデータを GROUP BY 句によってグループ化し SUM，MIN，MAX，および，COUNT の集約演算を行う処理である．

(ii) における“ステートレス”とは，アクセラレータの処理結果が，DB からの指示とストレージに格納されたデータのみで決定されることを意味する．“ステートフル”なアクセラレータの処理結果は，事前設定パラメータや先行処理結果などに依存するため，アクセラレータの状態管理の必要が生じる．これに対し，ステートレスなアクセラレータは，状態管理が不要という利点を有する．

(iii) における DMA（Direct Memory Access）とは，メモリとデバイス間のデータ通信を，CPU を介さずに DMA コントローラで実行する通信方式である．提案する FPGA アクセラレータは，サーバ装置と高速に通信するために DMA を用いる．

我々は，(i)(ii)(iii) の特性を有し，Hadoop システムの標準的なカラムストアフォーマットである Parquet 形式のデータを処理する HW 回路を開発した[*1]．この HW 回路は，[20] で説明した HW 回路に対して，Parquet 形式からデータを取り出すデコーダを追加することで開発した．

2.2 Apache Drill

Drill は，表 1 に示すように，User, Processing, Data Sources の三つのレイヤによって構成される [7]．User レイヤは，Java Database Connectivity（JDBC）や Open Database Connectivity（ODBC）などのインターフェースを提供するレイヤである．Processing レイヤは，データ処理を実行するレイヤであり，言語解析を実行する Parser，クエリプランを作成する Query Planner，クエリプランを実行する Execution Engine，データソースとのインターフェースを提供する Storage Engine によって構成される．Data Sources レイヤはデータを格納するレイヤであり，Hadoop Distributed File System（HDFS）などで構成される．

本稿は，SQL 処理の高速化を目的とすることから，User レイヤにおいて SQL 文を受信するシステムを対象にする．また，Hadoop システムの高速化を対象とすることから，Data Sources レイヤに HDFS を使用するシステムを対象にする．

3 システムアーキテクチャの提案

本章では，3.1 節でアーキテクチャ検討に用いた要件を示し，3.2 節で提案するシステムアーキテクチャを示す．

3.1 システムアーキテクチャの検討

開発効率と高性能化を両立するために以下の三つの観点からシステムアーキテクチャの要件を設定した．

(要件 1)　Drill の実装に依存する機能と依存しない機能の分離

[*1] http://www.hitachi.com/New/cnews/month/2017/11/171114.html

表 1　Apache Drill のアーキテクチャとコンポーネント

Layer	Example of Component	Function
User	JDBC/ODBC	Provide interface
	Command line interface	
	REST interface, etc.	
Processing	Parser	Parse language
	Query planner	Build plan
	Execution engine	Execute plan
	Storage engine	Provide interfaces with data stores
Data sources	HDFS	Store data
	Relational Databases	
	MongoDB, etc.	

(要件 2)　処理内容の開発言語に対する適合性

(要件 3)　データ処理を行う適切な演算装置の選択

(要件 1) は，開発効率の向上を目的とする要件である．オープンコミュニティで開発されている Drill に対しては頻繁にアップデートが行われる．頻繁なアップデートへの追従を容易にするため，Drill の実装に依存する機能と依存しない機能の分離実装が求められる．

(要件 2) は，処理性能と開発効率の向上を目的とする要件である．Drill は Java 言語で実装されている．一方，2.1 節に述べたように，提案する FPGA アクセラレータは，サーバ装置との通信に DMA を利用する．Java 言語のメモリ操作の制約（Java 仮想マシンのメモリ量の制約，確保可能な連続メモリ量の制約）から，Java 言語のメモリ空間への DMA は実装困難である．Java 言語で開発困難な機能に関しては，C 言語などのメモリを直接操作可能な言語を使用する必要がある．

(要件 3) は，処理性能の向上を目的とする要件である．近年，小規模な CPU が搭載されている FPGA ボードが多数販売されている．また，FPGA チップ内に Software CPU を構成することも可能である．アクセラレータシステムにおいては，サーバの CPU，FPGA の CPU，FPGA の HW 回路の三つの演算装置が存在し，高性能化のために，演算装置の特性に合わせた使い分けが求められる．

3.2 システムアーキテクチャの提案

本稿で提案するシステムアーキテクチャを図 1 に示す。提案するシステムアーキテクチャは，(1)–(5) の 5 個のコンポーネントで構成される．各コンポーネントが担う機能と (要件 1)–(要件 3) による機能分類を表 2 に示す．本節以降では，本アーキテクチャの各コンポーネントについて説明する．

3.2.1 Drill Plugin

Drill Plugin は，Drill の実装に依存する機能を担うコンポーネントである．本コンポーネントでは，表 2 に示す機能#1#2 および#8 が動作する．Drill Plugin は，Java で実装されている Drill と連携するため，開発言語には Java が適している．

#1 は Drill が生成した実行プランを FPGA アクセラレータに適合した実行プランに変換する機能である．本機能の詳細は 4.1 節で説明する．#2 は FPGA アクセラレータに指示を与えるた

表 2　統合機能と要件 1 – 3 による分類

#	統合機能	要件 1 Drill 実装依存	要件 2 開発言語適合性	要件 3 演算装置適合性
1	実行プランの変換	✓	Java	サーバの CPU
2	FPGA コマンドの生成	✓	Java	サーバの CPU
3	HDFS からのデータの読み出し		C	サーバの CPU
4	FPGA へのコマンドの送信		C	サーバの CPU
5	HW 回路の制御		C	FPGA の CPU
6	データ抽出処理		HDL	FPGA の HW 回路
7	FPGA からの処理結果の受信		C	サーバの CPU
8	Drill への処理結果の送信	✓	Java	サーバの CPU

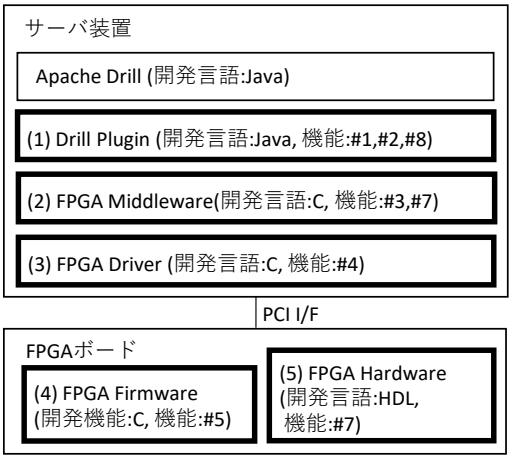

図 1　Drill と FPGA アクセラレータを統合するシステムアーキテクチャ

めの FPGA コマンドを生成する機能である．本機能は Drill の Processing レイヤにおいて，FPGA 向けの Storage Plugin として動作する．#8 は FPGA の処理結果を Drill に送信する機能である．Drill はデータを Value Vector と呼ばれる形式で管理しており，本機能では FPGA の処理結果を Value Vector 形式にして Drill に送信する．本機能は#2 と同様に，FPGA 向けの Storage Plugin として動作する．

3.2.2　FPGA Middleware

FPGA Middleware は FPGA アクセラレータとの通信を高速に行うためのコンポーネントであり，#3 および#7 の機能が動作する．DMA 通信を行うために連続したメモリ領域を確保する必要があり，FPGA Middleware の開発言語としては C 言語が適している．

#3 は SQL 処理に必要なデータを HDFS ファイルから読み出し，サーバ装置のメモリに格納する機能である．FPGA は DMA を用いてサーバ装置のメモリからデータを取得して処理を実行する．#7 は FPGA から処理結果を受信する機能である．FPGA の処理結果は DMA を用いてサーバ装置のメモリに格納される．FPGA Middleware は FPGA が生成した複数の処理結果をまとめて Plugin に送信する．

3.2.3　FPGA Driver

FPGA Driver は FPGA との通信を行うためにオペレーティングシステムの特権モードで動作するソフトウェアである．FPGA Driver，FPGA Middleware，および，FPGA アクセラレータ間の通信プロトコルに関しては，[20] において詳細を記述している．

3.2.4　FPGA Firmware

FPGA Firmware は HW 回路の起動・停止などの制御を行うために FPGA ボード内で動作するソフトウェアである．FPGA ボード内に設置された CPU，または，FPGA チップ内に構成された Software CPU で動作する．低消費リソースと高速処理が求められるため，C 言語での開発が適している．

3.2.5　FPGA Hardware

FPGA Hardware は 2.1 節に述べた Filter 処理と Aggregation 処理を実行する HW 回路である．これらの処理は，HW 回路での実行に適しており，我々は HW 回路を，Hardware Description Language（HDL）を用いて開発した．提案する FPGA アクセラレータが高速に処理を行うためには，圧縮辞書などのメタデータが FPGA の内部メモリに格納されている点が重要である [20]. Parquet は生成時のパラメータによってメタデータのサイズを調整可能であり，本研究では，メタデータが FPGA の内部メモリに納まるように Parquet ファイルの生成時のパラメータを設定した．

4　システムアーキテクチャの実装

本章では，3 章に挙げた機能のうち，実行プラン変換機能と，Drill Plugin と FPGA Middleware 間の通信方式について説明する．

4.1　実行プラン変換機能

DB は，Scan，Filter，Project，Aggregation などの実行プランのノードを順に実行することで SQL 文の処理を行う．FPGA アクセラレータに集約処理をオフロードするためには，これら Scan，Filter，Project，Aggregation を一つのノードにまとめる必要がある．

Drill は実行プランの生成に Apache Calcite（Calcite）を用いている [2]．図 2 に示すように，Calcite は実行プラン作成のためのルールを設定する機能を備える．Calcite は SQL Parser & Validator において，SQL 文を関係代数による表現に変換する．この関係代数表現に対して，Calcite の Query Optimizer は，Logical

Rules と Physical Rules を用いて，実行プラン（Physical Plan）を生成する．我々は，Calcite に設定するルールとして，表 3 に示す FPGA Rules を追加した．FPGA Rules には以下の (a)–(c) が記載されている．

(a) 変換前のノードの組
(b) 変換の可否判定条件
(c) FpgaScan のパラメータ生成方法

Calcite は，実行プランを走査して，(a) に記載のノードの組を検知すると，(b) の判定条件に従って，FpgaScan ノードへの変換可否を判定する．本判定は，検索条件数やカラム数などの，FPGA の機能制約に従って実施される．FpgaScan ノードへの変換が可能な場合，(c) に従って，変換前のノードの処理を，FPGA アクセラレータで実行するためのパラメータが生成され，FpgaScan ノードに設定される．表 3 の変換前のノードは，ストレージからのデータの読み出しを行う Scan ノードを含んでいる．そのため，ストレージからデータを読み出した直後の Filter 処理と Aggregation 処理が，FPGA アクセラレータの処理に変換される．Calcite による実行プランの変換は，適用可能なルールがなくなるまで，繰り返し実行される．

FPGA Rules の適用例を TPC-H の Query1 を例に説明する．図 3 左は FPGA Rules 適用前の実行プランである（各ノードに付与されている番号は，Drill によって生成された Operation ID である）．図 3 の 03-05, 03-06 のノードに表 3 のルール#1 が適用され，二つのノードを合わせて FpgaScan ノードが生成される．次に FpgaScan ノードと 03-03 の Project ノードに対してルール#4 が適用され，03-03 から 03-06 を合わせた FpgaScan ノードが生成される（SelectionVectorRemover は Drill のデータ管理形式である Value Vector 固有の処理であり無視するルールにしている）．最後に，FpgaScan ノードと 03-02 の HashAgg ノードにルール#2 が適用され，03-02 から 03-06 を合わせた FpgaScan ノードが生成される．

FPGA Rules の適用においては，FPGA アクセラレータの機能制約に従いチェックを行い，機能制約に違反する場合には実行プランの変換を行わない．これにより，FPGA アクセラレータで実行可能な処理だけがオフロードされ，FPGA アクセラレータで実行不可能な処理はソフトウェアで処理が行われる．

Drill の Execution Engine は，FpgaScan ノードを検知すると，対応する Storage Engine を呼び出して処理を行う．本研究では，FpgaScan の処理を行う Storage Engine を開発した．本 Storage Engine は，FpgaScan の処理内容を FPGA に送信する機能，および，FPGA の処理結果を Drill に送信する機能を提供する．これにより，Scan，Filter，Project，および，Aggregation のノードをまとめた FpgaScan ノードが一括して FPGA アクセラレータにオフロードされる．

4.2 Drill Plugin と FPGA Middleware 間の通信方式

3.2 節に述べたように，Drill Plugin は Java 言語に適合し，FPGA Middleware は C 言語に適合する．Java 言語と C 言語のプログラム間の通信には，Java Native Interface（JNI）が広く用いられている．

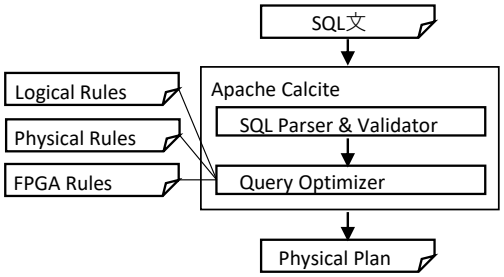

図 2 Apache Calcite による実行プランの生成フロー

表 3 Apache Calcite に設定した FPGA Rules

#	変換前のノード	変換後のノード
1	Filter–Scan	FpgaScan
2	Aggregation–Scan	FpgaScan
3	Aggregation–Project–Scan	FpgaScan
4	Project–Scan	FpgaScan

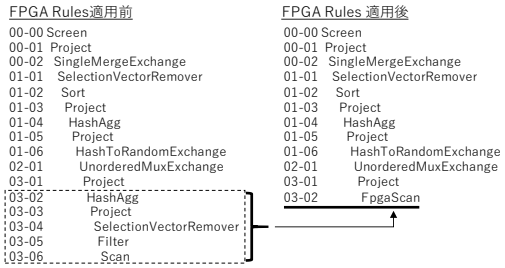

図 3 FPGA Rules の適用による実行プランの変化

JNI 通信のオーバヘッドを調べるため，表 4 の環境において，1 カラムを SELECT する SQL 文（SELECT l_partkey FROM lineitem WHERE l_partkey < 24;）と 2 カラムを SELECT する SQL 文（SELECT l_partkey, l_suppkey FROM lineitem WHERE l_partkey < 24;）を実行し，JNI 通信に要した時間を測定した．本測定では，JNI 通信のオーバヘッドに着目するため，ストレージからのデータの読み出しは行わず，FPGA Middleware から Drill Plugin に固定のデータを送信する改造を行って測定を行った．

JNI 通信オーバヘッドの測定結果を図 4 に示す．図 4 に示すように，応答行数の増加に比例して JNI 通信時間が増加する．また，SELECT するカラム数に比例して JNI 通信時間が増加している．本測定結果から，JNI 通信のオーバヘッド削減のためには，JNI 通信の回数削減が重要であることが判明した．

JNI 通信オーバヘッドを削減する方策として，図 5 に示すカラムバルク転送方式を採用した．カラムバルク転送方式では，FPGA アクセラレータの処理結果をもとに，カラムごとにデータをまとめて Drill Plugin に送信するカラムバルク送信機能を FPGA Middleware に設置する．また，カラムのデータから Drill のデータ形式（Value Vector）を生成する Value Vector 生成機能を Drill Plugin に設置する．

図 4 にカラムバルク転送方式の適用効果を示す．図 4 には 1 カラム SELECT の SQL に対して，100 個のデータをまとめて転

表 4　評価環境

Server	HA800 RS210
CPU	Xeon E5649, 2 Sockets
Memory	48GB (DDR3 1333MHz)
Operating System	CentOS 7.5
Java Version	1.8.0 161
Drill Version	1.10.0

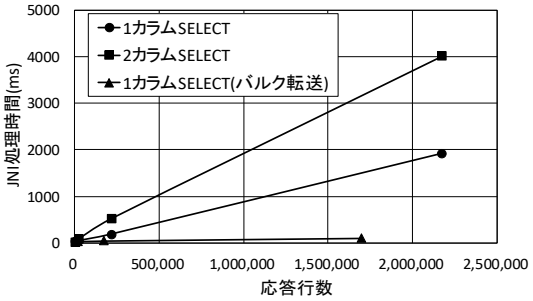

図 4　JNI 通信オーバヘッドとカラムバルク転送方式の適用効果

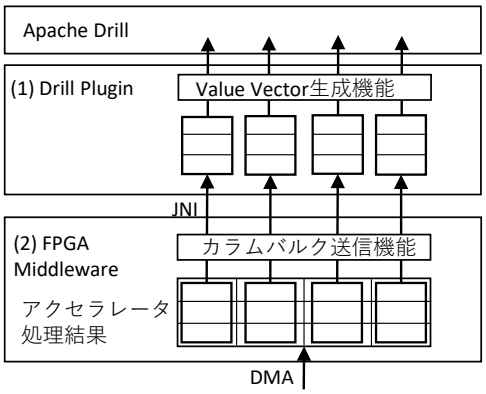

図 5　カラムバルク方式の模式図

送するカラムバルク転送方式を適用した結果を示している．図 4 に示すように，JNI 通信回数の削減によって，Java 言語と C 言語のプログラム間の通信オーバヘッドの削減が可能である．

大容量のデータ分析では，データ量の増大に伴い FPGA からの応答行数も増大することが想定される．図 4 の評価結果から，応答行数の増大に伴い大幅にシステムの性能が低下することが想定される．カラムバルク方式により，応答行数の増大に伴う性能低下が抑えられ，FPGA アクセラレータの大規模なデータ分析システムへの適用が可能になる．

5　Discussion

本章では，DB のアクセラレータシステムの実用化に向けたオープンプロブレムについて論じる．以下の 5.1–5.4 の項目は [5] に列挙されているオープンプロブレムからの引用である．

5.1　How to achieve line-rate data processing (what level of parallelism can be attained)?

提案するシステムアーキテクチャでは，FPGA に設定される HW 回路において，Data 並列化と Pipeline 並列化の手法を用いている [20]．また，2 章で述べたように，アクセラレータをステートレスに設計することで，複数のアクセラレータが並列に動作可能である．ステートレスな設計により，アクセラレータの状態管理が不要であり，Drill が有する並列化機構（クラスタ処理，スレッド処理，マルチコア処理）は，アクセラレータを適用しない場合と同様に動作可能である．

我々が提案する FPGA アクセラレータの処理性能は，HW 回路の 1Cycle Clock あたり，約 2 行から 0.5 行のデータを処理可能である（SQL 文の複雑度によって変動する）[20]．従って，FPGA の動作周波数を 100MHz とすると，200M 行/s から 50M 行/s の処理が可能である．一方，TPC-H の Lineitem 表 7.3GB を Parquet ファイルで格納した場合のデータサイズは 6.0GB であり，データ圧縮率は 0.82（= 6.0/7.3）であった．SQL 文で 8B の数値カラム 4 個を使用するケースを想定すると，1 行当たりのデータ量は 26.24B（$= 8 \times 4 \times 0.82$）である．従って，200M 行/s の処理では 5.2GB/s （$= 26.24 \times 200$M），50M 行/s の処理では 1.3GB/s（$= 26.24 \times 50$M）の性能で SSD からデータを読み出す必要がある．一方，1 台当たり 3.5GB/s 以上の読み出し性能を有する SSD が製品化されている．

以上の試算は，FPGA の動作周波数，SQL 文，データの圧縮率によって変化する．しかしながら，上記の試算から，データ抽出処理に関しては，おおよそ，SSD1 台に対して FPGA アクセラレータ 1 台で処理性能のバランスが取れると言える．

5.2　How to overcome the hardware inflexibility and development cost challenges?

提案するシステムアーキテクチャでは，データ分析において最初に実行されるデータ抽出だけをアクセラレータで実行し，その他の処理はサーバ装置で実行する．データ抽出は実行頻度が高く，HW 回路開発の投資対効果が高い．また，7 章に述べるように，Drill 以外の DB に対しても FPGA アクセラレータを適用することで，さらなる投資対効果の向上が見込まれる．

HW 回路はソフトウェアと比較して柔軟性が低く，実行できる処理には各種の制約が生じる（SQL 文で処理できるカラムの個数，WHERE 句の条件数など [20]）．この問題に対応するため，提案するシステムアーキテクチャでは，FPGA Rules において，HW 回路での実行可否を判定し，HW 回路で実行できない処理はソフトウェアで実行する．すなわち，提案するシステムアーキテクチャでは，HW 回路の低柔軟性に対応するために，アクセラレータとサーバ装置に処理を振り分ける機構を設置している．

5.3　How/Where to place hardware accelerators in query execution pipelines in practice?

データ抽出処理をアクセラレータで実行する場合，図 6 左に示すように，FPGA に処理内容を指示するコマンド送信機能，および，FPGA の処理結果を受信するデータ受信機能を DB に追加する必要がある．我々が提案するシステムアーキテクチャでは，Calcite の Rule Plugin 機構と Drill の Storage Plugin 機構を用い

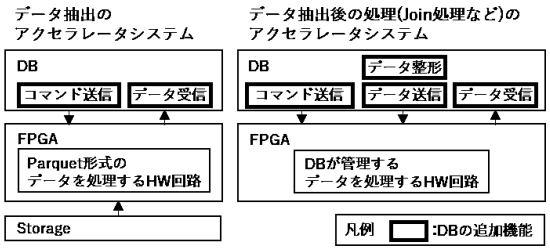

図 6 データ抽出処理とデータ抽出後の処理にアクセラレータを適用する場合の相違

て，Drill のソースコード改変なしに，これらの機能を実装した．

HW 回路開発の投資対効果の観点からは，データ抽出後に実行される Join 処理なども実行頻度が高く，投資対効果が高い．しかしながら，図 6 右に示すように，データ抽出後の処理にアクセラレータを適用する場合には，コマンド送信機能とデータ受信機能に加えて，DB が管理するデータをアクセラレータに移動するためにデータ送信機能が必要である．DB が管理するデータには，CPU での処理を高速化するための各種の最適化が施される [3, 11]．そのため，単純なデータの移動ではなく，FPGA で処理を行うためのデータ整形の必要も生じる．

DB が管理するデータを送受信するためのフォーマットが Apache Arrow（Arrow）において検討されている[*2]．DB が Arrow に対応することで，DB が管理するデータの入出力が容易になり，データ抽出後の処理に対するアクセラレータの適用が促進されると期待される．しかしながら，アクセラレータの効果を得るためには，データ入出力のオーバヘッドを上回る高速化が必要である．一方で，ストレージから DB へのデータ移動は，アクセラレータの適用有無に関わらず必要な処理である．また，FPGA アクセラレータが SQL 文で使用するデータを抽出するため，DB がデータ受信に要する処理負荷が削減される．

以上に述べたように，アクセラレータの適用により生じる，DB の改造工数，および，処理オーバヘッドの観点を鑑みると，データ抽出はアクセラレータの適用に適した処理であると言える．

5.4 What are the power and energy consumption benefits of hardware acceleration?

サーバ装置に設置可能な DRAM 容量の制限から，大容量データをインメモリデータベースで処理するためには，多数台のサーバ装置が必要である．一方，SSD には数 TB のデータを格納可能であり，サーバ装置 1 台に複数台の SSD を接続可能である．我々は先行研究 [20] において，FPGA アクセラレータの適用により，SSD にデータを格納した場合にも，インメモリデータベースと同等のデータ抽出性能が得られることを示した．サーバ台数の削減は，電力消費量を大幅に削減する．1 章に述べた DRAM の SSD 置き換えだけでなく，サーバ台数削減による電力消費量削減の効果が見込まれる．

Hadoop システムでは，大容量データの高速分析ために，多数のサーバ装置によるインメモリクラスタが用いられる．しかしながら，サーバ装置間のネットワークオーバヘッドのために，Join 処理の性能はサーバ装置の台数に対してスケールしないことが示されている [10]．サーバ台数の削減によりネットワークオーバヘッドが削減されるため，FPGA アクセラレータは，Join 処理の高速化に対しても効果を有する可能性がある．以上に説明した，サーバ台数削減による電力消費量の削減，および，Join 処理の高速化に関しては，今後，検証を進める計画である．

6 関連研究

6.1 DB とアクセラレータの統合方式

[14] において，User Defined Function（UDF）を用いて MonetDB と FPGA アクセラレータを統合し，正規表現検索，スカイライン関数，Stochastic Gradient Decent 関数などを高速化する doppioDB が提案されている．多くの DB では，タプルごとに UDF が呼び出されるため，UDF の呼び出しオーバヘッドにより性能が低下する．これに対し，MonetDB の UDF では一度の呼び出しで多数のタプルを処理可能であり，UDF の呼び出しオーバヘッドが小さい [14]．UDF を用いる統合方式のデメリットとしては，SQL 文の変更が必要な点がある．UDF を用いる方式では，FPGA にオフロードするための関数を設定し，SQL 文に関数名を指定する必要がある．これに対し本研究では，FPGA で実行する処理を Plugin で抽出することにより，SQL 文の変更なしに，Drill と FPGA アクセラレータの統合を実現している．

DB と FPGA アクセラレータを統合する別方式としては，DB の外部データアクセス機能やストレージエンジン機能を用いる方式がある．Swarm64DB は，PostgreSQL の Foreign Data Wrapper 機能，MariaDB および MySQL の Storage Engine Interface 機能を用いて，FPGA に処理をオフロードする [16]．これらの機能では，SQL 文の実行プランの変更は行わないため，FPGA にオフロードできる処理の範囲に制約が生じる．PostgreSQL の Foreign Data Wrapper 機能でオフロードできるのはフィルタ処理だけであり，集約処理のオフロードはできない [6]．また，[21] では MySQL の Storage Engine Interface 機能に変更を加えて，集約処理のオフロードを実現している．

6.2 アクセラレータシステムの設計

[5, 13] において，アクセラレータシステムの分類が提示されている．本分類に従うと，提案するシステムアーキテクチャは，system レベルは co-processor，programming レベルは hardware description languages, representational レベルは parameterized circuits，algorithmic models レベルは data and pipeline parallelism に分類される．

system レベルに関しては，DB はデータ検索やトランザクション処理など多様な機能を備え，DB の全機能の FPGA での実行は現実的ではない．そのため，DB システムに対するアクセラレータの適用においては，CPU とアクセラレータが協業する co-processor は妥当性の高い設計であると言える．

programming レベルに関しては，OpenCL や CUDA など有望な選択肢が出現している．OpenCL や CUDA で生成した HW 回路は，高いスキルの技術者が HDL で作成した HW 回路に対して

[*2] https://arrow.apache.org/

性能面で劣る．しかしながら，開発効率の面では優位であることから，高位合成技術の向上に伴い，今後は OpenCL や CUDA を用いた開発の優位性が向上していくと想定される．

representational レベルに関しては，SQL 文に応じて HW 回路を動的に入れ替えるシステムの研究が行われている [12, 19, 22]．SQL 文に応じた HW 回路の入れ替えは有望な技術であり，HW 回路の動作をパラメータで指定する parameterized circuits と比較して数倍から数十倍の性能向上が見込まれる．しかしながら，FPGA の HW 回路の入れ替えには 1 秒から 2 秒を要する [19]．また，我々の開発環境においては，HW 回路の入れ替えの失敗が数十回から数百回に 1 回の割合で発生した．そのため，現行技術において，SQL 文に応じた HW 回路の入れ替えは実用的でないと判断し，本研究では，parameterized circuits を用いた．

algorithmic models レベルに関しては，Data parallelism と Pipeline parallelism は，FPGA に設置する HW 回路の性能向上のための主要技術である [17]．FPGA に設定する HW 回路にこれらの技術を適用することで，FPGA に搭載されるロジックエレメント数に応じた性能向上が可能である．DB システムの FPGA アクセラレータにおいては，data and pipeline parallelism は妥当性の高い設計であると言える．

7 結論と今後の課題

FPGA アクセラレータは，アクセラレータを適用しない場合と比較して，処理性能と消費電力を大幅に向上する可能性を有している．本稿では，オープンソースのデータベースである Apache Drill と FPGA を用いたアクセラレータを統合するシステムアーキテクチャを提案した．アーキテクチャ設計に用いた要件を提示し，本要件にもとづくシステムアーキテクチャの設計を行った．提案するシステムアーキテクチャは，Drill のソースコード改変が不要，Drill と FPGA アクセラレータの高速なデータ通信，という二つの特徴を有する．また，本稿では，提案したシステムアーキテクチャによるオープンプロブレムの解決方法について論じた．

Parquet は Hadoop システムの標準的なカラムストア形式であり，Drill 以外の DB においても使用されている．また，本提案のアーキテクチャにおいて，Drill の実装に依存する機能は Plugin のみに実装されており，Plugin の開発だけで他の DB に適用可能である．他の DB 向けの Plugin を開発し，FPGA アクセラレータを適用可能な DB を拡大することが今後の課題である．

参考文献

[1] Daniel J Abadi, Samuel R Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 967–980. ACM, 2008.
[2] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J Mior, and Daniel Lemire. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *Proceedings of the 2018 International Conference on Management of Data*, pages 221–230. ACM, 2018.
[3] Peter A Boncz, Martin L Kersten, and Stefan Manegold. Breaking the memory wall in monetdb. *Communications of the ACM*, 51(12):77–85, 2008.
[4] Philippe Bonnet. What's up with the storage hierarchy? In *CIDR*, 2017.
[5] Rajesh R Bordawekar and Mohammad Sadoghi. Accelerating database workloads by software-hardware-system co-design. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1428–1431. IEEE, 2016.
[6] The PostgreSQL Development Group. Postgresql 11.1 documentation,f.33.postgres_fdw. `https://www.postgresql.org/docs/11/postgres-fdw.html`, 2018.
[7] Michael Hausenblas and Jacques Nadeau. Apache drill: interactive ad-hoc analysis at scale. *Big Data*, 1(2):100–104, 2013.
[8] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, et al. Bluedbm: Distributed flash storage for big data analytics. *ACM Transactions on Computer Systems (TOCS)*, 34(3):7, 2016.
[9] Sungchan Kim, Hyunok Oh, Chanik Park, Sangyeun Cho, and Sang-Won Lee. Fast, energy efficient scan inside flash memory ssds. In *Proceeedings of the International Workshop on Accelerating Data Management Systems (ADMS)*, 2011.
[10] Kunal Vikas Kulkarni. *Performance Characterization and Improvements of SQL-On-Hadoop Systems*. PhD thesis, The Ohio State University, 2016.
[11] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment*, 3(1-2):330–339, 2010.
[12] Mohammadreza Najafi, Mohammad Sadoghi, and Hans-Arno Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks. In *2015 IEEE 31st International Conference on Data Engineering*, pages 819–830. IEEE, 2015.
[13] Mohammadreza Najafi, Kaiwen Zhang, Mohammad Sadoghi, and Hans-Arno Jacobsen. Hardware acceleration landscape for distributed real-time analytics: Virtues and limitations. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1938–1948. IEEE, 2017.
[14] David Sidler, Zsolt István, Muhsen Owaida, Kaan Kara, and Gustavo Alonso. doppiodb: A hardware accelerated database. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1659–1662. ACM, 2017.
[15] Mike Stonebraker, Daniel J Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, et al. C-store: a column-oriented dbms. In *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker*, pages 491–518. 2018.
[16] swarm64. Swarm64db feature documentation. `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/third-party/swarm64db-solution-sheet.pdf`, 2017.
[17] Jens Teubner and Louis Woods. Data processing on fpgas. *Synthesis Lectures on Data Management*, 5(2):1–118, 2013.
[18] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 231–242. ACM, 2010.
[19] Zeke Wang, Johns Paul, Hui Yan Cheah, Bingsheng He, and Wei Zhang. Relational query processing on opencl-based fpgas. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–10. IEEE, 2016.
[20] Satoru Watanabe, Kazuhisa Fujimoto, Yuji Saeki, Yoshifumi Fujikawa, and Hiroshi Yoshino. Column-oriented database acceleration using fpgas. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 686–697. IEEE, 2019.
[21] Louis Woods, Zsolt István, and Gustavo Alonso. Ibex: an intelligent storage engine with support for advanced sql offloading. *Proceedings of the VLDB Endowment*, 7(11):963–974, 2014.
[22] Daniel Ziener, Florian Bauer, Andreas Becher, Christopher Dennl, Klaus Meyer-Wegener, Ute Schürfeld, Jürgen Teich, Jörg-Stephan Vogt, and Helmut Weber. Fpga-based dynamically reconfigurable sql query processing. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 9(4):25, 2016.