# 複数クエリ統合によるDFDから ソフトウェアコンポーネントへの変換手法

木村 功作[1]　野村 佳秀[1]　栗原 英俊[1]　山本 晃治[1]　山本 里枝子[1]

**概要**：本稿では，DFD 中の複数のプロセスを統合することで少数のソフトウェアコンポーネントに変換する手法を提案する．大規模イベントデータ処理技術を容易に利用するために DFD を提供する開発支援環境では，DFD 中のプロセスを実現するクエリと同じ数だけ多くのソフトウェアコンポーネントが生成される．このような環境では，大量のデータを処理する時にソフトウェアコンポーネント間のデータ送受信のオーバーヘッドが大きいという問題がある．提案手法では，同時に処理可能な複数のクエリを一つのクエリに統合する手法と，前段処理のクエリをサブクエリとして後段処理のクエリに埋め込むことで処理の順序を維持しつつ統合する手法を用いてクエリを統合する．提案手法によって，データの送受信を含めた実行時間が削減されることが実験によって確認出来た．

# A Method of DFD to Software Components Transformation using Multi-query Integration

Kosaku Kimura[1]　Yoshihide Nomura[1]　Hidetoshi Kurihara[1]　Kouji Yamamoto[1]
Rieko Yamamoto[1]

***Abstract:*** This paper proposes a method that transforms few software components from DFD integrating multiple processes. The development environment which provides DFD to facilitate utilizing technologies for massive event data processing generates software components as many as queries which represent processes in DFD. Such environments have a problem that processing massive data causes the big overhead of data transmission between software components. The experimental results show that the proposed method can reduce the execution time comprising data transmission time.

## 1. はじめに

MapReduce[6] や複合イベント処理 (Complex Event Processing: CEP) に代表される大規模イベントデータ処理技術は，スマートフォンやタブレット端末等のデバイスやセンサ，または日常的に利用している Web サービスやその他 ICT システムが生成する大量のイベントデータを効率的に処理することが可能である．今まで蓄積されるだけで活用されていなかった，または今後生成されるであろう大量のイベントデータから新たな価値を創出するために，これらの大規模イベントデータ処理技術を実装技術の知識を持たないユーザでも容易に活用可能な開発環境の整備が重要となっている．

大規模イベントデータ処理は，データや処理内容の性質から蓄積型，ストリーム型の二種類に分類される．前者は，従来のデータベース管理システム (DBMS) を用いたバッチ処理のように，蓄積したデータに対して逐次的に処理を行うものであり，大量のデータを扱えるように並列分散化した技術の一例として MapReduce がある．後者は，データの抽出・加工等の条件を予め登録しておき，継続的に流れてくるデータをリアルタイムに処理するものであり，CEP やストリームデータ処理，データストリーム管理システム (DSMS) 等と呼ばれる．これらの処理における処理手順は，種々の Business Intelligence ツールや ETL ツール[*1]等で用いられるデータフロー図 (DFD) で表されるのが直観的である．

大規模イベントデータ処理の既存の開発環境としては，CEP における継続的クエリをフローで記述させるも

[1] 富士通研究所　ソフトウェアイノベーション研究部
Software Innovation Lab., Fujitsu Laboratories Ltd.

[*1] Extract/Transformation/Load．データウェアハウスにおいてデータを利用しやすい形に抽出・加工するツール

の [1], [2], [3] や，MapReduce 処理を DFD や簡単な DSL で記述させるもの [10], [13] 等がある．しかし，これらは単体の技術についてのみ支援するものであるため，データ分析やサービスのためのリアルタイム処理を行うにはそれぞれ異なる開発環境を用いる必要がある．そのため，我々は前述の蓄積型，ストリーム型の複数の大規模イベントデータ処理技術を統合的に支援することを目的として，DFD を用いた大規模イベントデータ分析開発支援環境の研究開発を行っている [15]．本環境は，SQL のようなクエリ言語によってデータの処理を定義するような処理エンジンを対象とした開発支援を行うものである．

本環境では，DFD 中の殆どのプロセスは，それぞれ適切な処理エンジンでクエリを処理するような実装に変換される．このような環境では，蓄積型やストリーム型の処理エンジンはそれぞれ異なる非機能要件を持つため，DFD 中のプロセスの実装はそれぞれ異なる適切な構成の実行環境に配備されるべきである．そこで，DFD 中のプロセスをソフトウェアコンポーネント (以後，コンポーネントと呼ぶ) に変換し，コンポーネント毎に異なる適切な実行ノードに配備し，コンポーネント間でデータ送受信を行うことで実行する．それぞれのプロセスは独立した処理であり，独立したクエリに変換可能なため，従来は一つのプロセスにつき一つのコンポーネントが生成されていた．しかし，従来手法では DFD が大きくなるほどコンポーネント数が多くなり，コンポーネント数が多くなるほどデータの送受信回数が多くなるため，処理するデータが大量になると，データ送受信のオーバーヘッドが無視出来ない程に大きくなるという問題がある．

この問題を解決するため，本稿では DFD 中の複数のプロセスを統合することで生成されるコンポーネント数を削減する手法を提案する．提案手法では，同時に処理可能な複数のクエリを一つのクエリに統合する手法と，前段処理のクエリをサブクエリとして後段処理のクエリに埋め込むことで処理の順序を維持しつつ統合する手法を用いてクエリを統合する．提案手法によって統合コンポーネントを生成することで，データの送受信を含めた実行時間が削減されることが実験によって確認出来た．

本稿の構成は以下の通りである．2 節で大量イベントデータ分析開発支援環境の概要について述べ，3 節で二種類の複数クエリ統合手法について述べ，4 節で提案手法の評価について述べ，5 節で関連研究について述べ，6 節でまとめる．

## 2. 大規模イベントデータ分析開発支援環境

大規模イベントデータ分析開発支援環境は，鉄道や POS 等のシステムで発生・蓄積している大規模イベントデータを元にした新たなサービスを実現するためのデータ分析，サービス開発，サービス評価を統合的に支援する環境である．本環境の主なユーザとして，分析のノウハウはあるが大規模イベントデータ処理のための実装技術に詳しくない分析者を想定している．分析者は，本環境上で処理手順を実装技術に依存しない DFD で定義することで，CEP や MapReduce 等の実装技術を実際に扱うこと無く大規模イベントデータの分析処理や，サービス実施のためのリアルタイム処理の開発が可能となる [15].

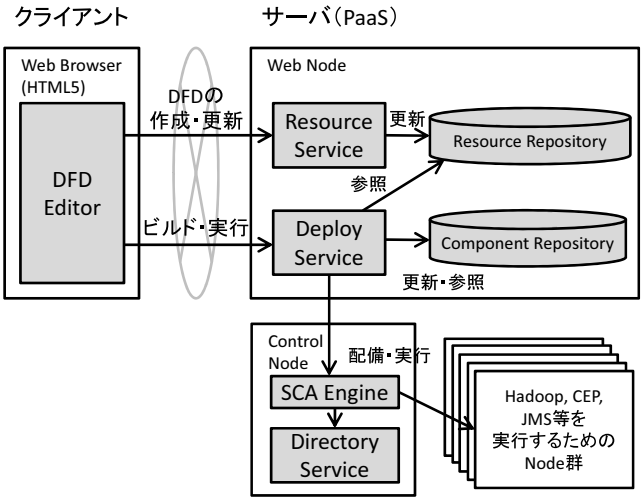


**図 1** 本環境の構成
**Fig. 1** Configuration diagram of the environment

本環境の構成を**図 1** に示す．Web Browser は分析者が所有する PC 上の Web ブラウザを指す．DFD Editor はその Web ブラウザ上で動作し，Web Node，Control Node，および各種処理エンジンを実行するための Node 群は全てパブリックまたはオンプレミスのクラウド上で動作することを想定している．分析者は，DFD Editor によってイベントデータの分析処理やサービス実施のためのリアルタイム処理の DFD を定義する．DFD は，Web Node 上の Resource Repository で管理され，Deploy Service によってビルド (コンポーネントに変換) される．変換されたコンポーネントは Component Repository で管理され，Deploy Service によって Control Node 上で実行される．Control Node は，コンポーネントの定義に従って Hadoop や CEP 等の処理エンジンやデータ受渡しに用いるキュー (Java Message Service: JMS) 等を実行するためのノードを配備し，イベントデータの分析処理やサービス実施のためのリアルタイム処理を行う．

本環境では，コンポーネントは Service Component Architecture (SCA)[*2]によって定義される．また，図 1 の Control Node でのコンポーネントの実行エンジンとして Apache Tuscany[*3]を用いている．SCA の定義に従い，各コンポーネントは Java 等で記述された実装と，指定された方式によって他のコンポーネントの実装との間でデータ

[*2] http://www.oasis-opencsa.org/sca
[*3] http://tuscany.apache.org/

の受け渡しを行うためのインターフェイスを持つ．各コンポーネントは，蓄積型，ストリーム型，それぞれの大規模イベントデータ処理のエンジンを用いてデータの処理を行う．本環境では，クエリを処理するエンジンとして蓄積型では Apache Hive[*4]，ストリーム型では Esper [*5]を用いている．

### 2.1 DFD による処理手順の定義

本環境では，データ分析とサービス開発における処理手順を DFD で定義する．データ分析では過去に蓄積されたイベントデータを MapReduce 等によって並列分散処理するのに対し，サービス開発では発生したばかりのイベントデータを CEP によってリアルタイム処理するように，両者はデータや処理の性質が大きく異なる．しかし，両者を共通の記法で定義することで，データ分析によって得られたルールをサービス開発に容易に反映することが可能となる．

本環境では，DFD におけるデータとそれに対するプロセスの依存関係から成り立つという性質は一般的な定義 [7] と同様であるが，データ源泉・吸収先と一時的な保管先としてのデータストアは区別せず，単にデータとして扱う．すなわち，本環境の DFD は互いに素な 2 つの集合，プロセスの集合とデータの集合があって，この 2 つの集合の要素間にのみ有向辺がある有向二部グラフである．

分析者が DFD Editor 上で定義した DFD は, XML Metadata Interchange (XMI)[*6]表現に変換され Web Node に送信，保存される．

プロセスの種類には，データを条件でフィルタ処理やフィールドの値を基に計算する処理，クラスタリング分析や多次元分析を行うもの，データのクレンジング処理やクーポン発行等のアプリケーションを提供する処理等，様々なものがあり，これらを DFD で組み合わせることでデータ分析やサービス開発の処理の流れを定義する．

プロセスには処理パラメータとしてのプロパティが，データにはフィールド名と型からなるスキーマとデータ蓄積方式が定義される．さらに，プロセスとデータの両方ともに処理型（ストリーム型または蓄積型）が定義される．

### 2.2 DFD からコンポーネントへの変換

Web Node は，DFD Editor からの分析者のリクエストによって XMI 表現にて保存された DFD の情報を基にコンポーネントへの変換を行う．Web Node は，プロセスの各種類，各処理型に対応したテンプレートを保持しており，プロセスのプロパティの値を引数としてテンプレートに適用することでコンポーネントの定義を生成する．

[*4] http://hive.apache.org/
[*5] http://www.espertech.com/
[*6] http://www.omg.org/spec/XMI/

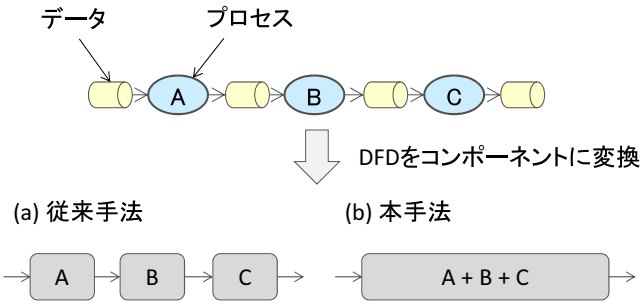


**図 2** DFD からコンポーネントへの変換
**Fig. 2** DFD to software components transformation

テンプレートには，クエリを生成するもの (クエリテンプレート) とコンポーネントを生成するもの (コンポーネントテンプレート) の二種類がある．クエリを処理することで実現可能なプロセスには，それに対応するクエリテンプレートが存在する．これらのプロセスは，クエリ処理エンジンに対応した汎用のコンポーネントテンプレートに，生成したクエリを引数として適用することでコンポーネントに変換される．一方で，クエリ処理以外の方法で実現されるプロセス (e.g. クラスタリング分析，多次元分析) は，それに対応する独自実装のコンポーネントテンプレートによって直接コンポーネントに変換される．

Web Node は，XMI 表現による DFD の定義とコンポーネント・クエリのテンプレート，変換されたコンポーネントをリポジトリで管理する．DFD Editor からの分析者のリクエストによって，Web Node はコンポーネントを実行環境へ配備し実行する．

多くの種類のプロセスをコンポーネント化した際の接続方法としては，キューまたは HDFS 等の上でのファイルによる受渡しが一般的であり，汎用性も高い．

## 3. 複数クエリ統合

従来手法では，DFD をコンポーネントに変換すると**図 2**(a) のようにプロセスと同じ数だけコンポーネントが生成される．この場合，分析処理やサービスのリアルタイム処理の実行時間にはコンポーネント間のイベントデータの送受信に要する時間が含まれるため，実行時のコンポーネント数が多く，かつ処理するイベントデータの件数が多いほど総実行時間におけるデータの送受信に要する時間の割合が多くなる．

この問題を解決するため，Esper や Hive 等，同一種類の処理エンジンでクエリ処理を行うプロセスのクエリを統合し，図 2(b) のように一つのコンポーネントにまとめて処理を行う．本手法では，句分割による統合と入れ子による統合の二種類の異なる統合手法を用いて統合を行う．

本節ではまず，句分割による統合と入れ子による統合について述べ，その後，これらの手法を用いる際の条件について述べる．

表 1　関係代数に基づくプロセスの分類
**Table 1**　Classification of processes based on the relational algebra

| 演算 | 順序維持 | SQL 記述例 | 説明 |
|---|---|---|---|
| 射影 | 不要 | `SELECT a, b, c FROM Table` | 特定のフィールドのみを抽出 |
| 選択 | 不要 | `SELECT * FROM Table WHERE 10 < a AND a < 20` | 特定の条件を満たすレコードのみを抽出 |
| 拡張 | 一部要 | `SELECT a, b, a + 2 * b AS c FROM Table` | 新しいフィールドを既存のフィールドから計算して追加 |
| 要約 | 要 | `SELECT COUNT(*) FROM Table` | 全レコードを要約した値（合計値，平均値等）を計算 |
| 複数入力 | 要 | `SELECT * FROM TableA JOIN TableB ON TableA.a = TableB.a` | 複数のデータを結合 (和，差，交差，直積，自然結合，左外結合等) |

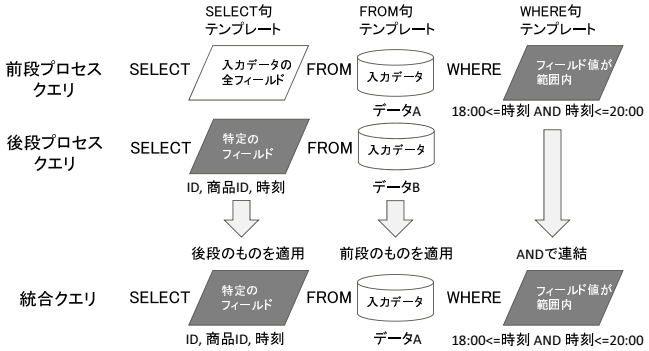


図 3　句分割による統合
**Fig. 3**　Query integration by clause separation

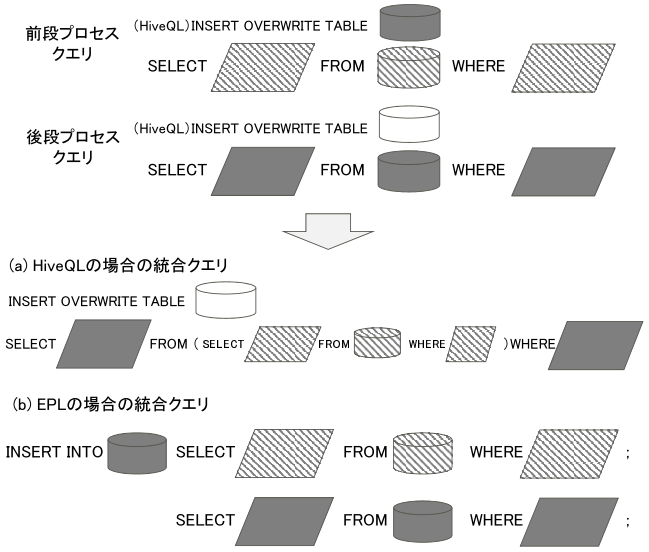


図 4　入れ子による統合
**Fig. 4**　Query integration by nesting

### 3.1　句分割による統合

句分割による統合は，同時に処理可能な複数のクエリを，SQL における `SELECT` 句等のクエリの句ごとに分割されたテンプレート (句分割テンプレート) を用いて，句ごとに分けて統合を行う手法である．句分割による統合では，DFD 中のプロセスとその後段プロセスについて，それぞれの句分割テンプレートを適用し各句の文字列を生成した後，単純な連結や置換によって複数のプロセスのクエリを句ごとに統合し，最後に句同士を連結することで統合クエリを生成する．その後，その統合クエリとさらに 1 つ後段のプロセスのクエリと統合し，これを繰り返すことで 3 個以上のクエリの統合を行う．

句分割による統合の例を**図 3** に示す．図 3 の場合，`SELECT` 句は後段プロセスのものを，`FROM` 句は前段プロセスのものをそのまま適用し，`WHERE` 句は前段プロセス，後段プロセスのものを `AND` で連結するのみで統合出来る．句分割による統合を行うため，各プロセスについて句分割テンプレートを予め作成しておく必要がある．

### 3.2　入れ子による統合

入れ子による統合は，前段プロセスのクエリを後段プロセスのクエリのサブクエリとして埋め込むことで，処理の順序を維持しつつ統合を行う手法である．入れ子による統合の例を**図 4** に示す．

Hive では，図 4(a) のように後段プロセスのクエリの `FROM` 句に記されているテーブルを前段プロセスのクエリに括弧を付けたものに置き換えることで埋め込む．一方，Esper では Hive のようなクエリを入れ子にする文法が認められていないため，図 4(b) のように前段プロセスのクエリを用いて `INSERT INTO` 句によって内部ストリームを作成し，後段プロセスのクエリの `FROM` 句に記されているテーブルを内部ストリームで置き換えることで埋め込む．入れ子による統合では，句分割による統合と同様に 1 回につきクエリを 2 個ずつ統合していくことで 3 個以上のクエリを統合する．

### 3.3　それぞれの手法を用いる際の条件

句分割による統合を行う際，DFD で定義された処理の順序に従って統合前の各クエリを逐次的に処理した結果と統合クエリの処理結果が異なることが無いように，DFD 中のプロセスのクエリを同時に処理可能なものとそうでないものに分類する必要がある．

本環境で用いる処理エンジンである Esper，Hive は，それぞれ Event Processing Language (EPL)，HiveQL というクエリ記述のための DSL を持っており，双方とも SQL に類似した構文となっている．SQL は関係モデルで表現されるテーブルに対する演算体系を表す関係代数をほぼ忠実に実装した言語であり，SQL に類似した EPL や HiveQL

のクエリは，SQL と同様に，関係代数における基本的な演算およびその応用的な演算によって分類することが出来る．本環境で考慮する関係代数の演算の一覧を**表 1** に示す．本環境では，二つのテーブルの和，差，交差，直積や各種結合は，全て複数の入力を結合する演算として共通に扱う．また，関係代数におけるその他の演算に属性名変更があるが，本環境では属性名変更を拡張の一部として扱う．

順序を維持しなければならない処理とは，処理の順序を入れ替えた時に出力データが異なるもの，または処理が成立しなくなるものを指す．表 1 の中で順序を維持しなければならない演算は次の通りである．

**複数入力**

複数の入力データを結合する処理は，各入力データに対する前段処理と順序を維持すべきである．なぜならば，各入力データと結合後の出力データはテーブルのスキーマやレコードの内容が変更されるため，結合後の出力データを前段処理の内容で処理すると結果が異なるからである．同様の理由で，結合後の後段処理とも順序を維持すべきである．

**要約**

合計値，平均値等の要約を算出する処理は，その前後の処理と順序を維持すべきである．なぜならば，入力データを他の処理によって内容が変更されたものに置き換えると算出結果が異なる場合があり，入力データの内容を変更すべきでないからである．

**拡張のうち，新規追加したフィールドが後続処理で参照されているもの**

このような処理は，その後続処理と順序を維持すべきである．なぜならば，処理の順序を入れ替えると，前段になった後続処理の入力データに参照すべきフィールドが存在しないので，計算・比較が出来なくなるからである．

句分割による統合は順序を維持する必要のないプロセス，すなわち，上記以外の演算 (射影，選択，拡張のうち，新規追加したフィールドが後続処理で参照されていないもの) のみを対象に実施する．なお，プロセスが複数の演算を含み，そのいずれかの演算が順序を維持しなければならない場合，句分割による統合は実施することが出来ない．

さらに，下記のようなプロセスは，句分割による統合でも入れ子による統合でも統合することが出来ない．

**前段処理の出力データを複数の処理が入力として用いるもの**

同一の入力データについての異なる処理を並行して行うものは，一つのクエリで記述することが出来ないため，クエリを統合することは出来ない．

**前後処理で処理型が異なるもの**

処理型が異なると，使用する処理エンジンの違いによりDSL が異なり，さらに入力データの性質（ストリームデータまたは蓄積データ）も異なるため，クエリを統合することは出来ない．

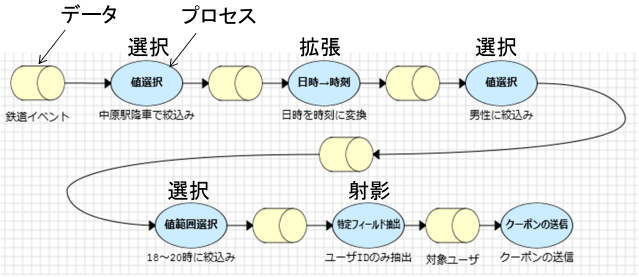


**図 5** クーポン発行サービスの DFD
**Fig. 5** DFD of a coupon service

本手法を実施するにあたり，プロセスの統合可否を事前に判定する必要がある．これは，XMI 表現による DFD の定義中のプロセスに入出力されるデータの数やスキーマ，プロセスのプロパティ，およびプロセスの `SELECT` 句と `WHERE` 句の句分割テンプレートを参照することで容易に実施することが出来る．

本手法では，順序を維持する必要の無いクエリについては句分割による統合によって同時に処理するように統合した後，句分割による統合で生成されたクエリを含めて，統合可能な全てのクエリを対象に入れ子による統合を実施するか，句分割による統合を事前に行わずに入れ子による統合のみ実施するかのどちらかを選択することが出来る．Hive と Esper の例のように，サブクエリの扱い方は処理エンジンによって異なり，処理エンジンによっては入れ子による統合の前に句分割による統合を実施することで実行時間が増えてしまう事が考えられる．そのため，処理エンジンごとに句分割による統合の実施するか否かを判断する必要があるが，これに関する議論は，4 節にて実験結果を用いて行う．

## 4. 評価実験

前節で述べた複数クエリ統合手法を実施することで，DFD をコンポーネントに変換して実行した際にデータ送受信を含めた実行時間がどれほど削減されるかを確認するため，仮想事例の DFD を用いて実験を行った．

本実験で用いた仮想事例である，鉄道の IC カードにおける改札等のイベントデータによるクーポン発行サービスの DFD を**図 5** に示す．図 5 の DFD 中のプロセスは，最後尾のプロセス「クーポンの送信」以外，いずれもストリーム型と蓄積型の両方の処理型で変換可能なものである．プロセス「クーポンの送信」は，入力されたイベントデータのユーザ ID フィールドを参照し，それに紐付けられたメールアドレスにクーポンを送信するアプリケーションである．本実験では，全プロセスの処理型をストリーム型，蓄積型のどちらかに統一し，ストリーム型では 10000

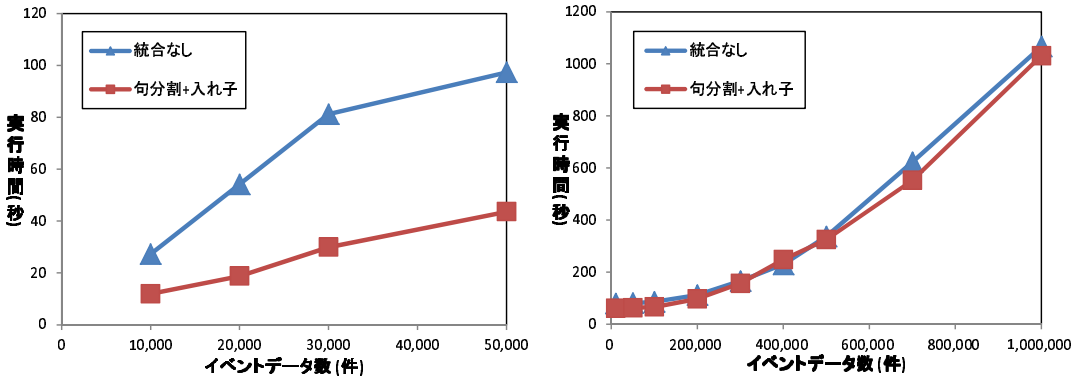


**図 6** 句分割による統合，入れ子による統合の二段階の統合を行った場合の実行時間

**Fig. 6** Execution time of the components integrated both by caluse separation and by nesting

件から 50000 件までの 5 種類のデータセット，蓄積型では 10000 件から 1000000 件までの 9 種類のデータセットを用いて，複数クエリ統合を用いる場合と用いない場合の双方でコンポーネントに変換し実行した際の実行時間を測定した．コンポーネントの初期化に要する時間を実行時間から除外するため，ストリーム型では DFD 中の最初のキュー「鉄道イベント」に全イベントデータを予め溜めておき，最後尾のプロセスに最初にイベントデータが到達した時刻から最後に到達した時刻まで，蓄積型では，最初のプロセスの処理開始時刻から最後尾のプロセスの処理終了時刻までを実行時間として測定している．

本実験では，DFD の定義をクエリやコンポーネントに変換するために Apache Velocity を採用し，図 5 の DFD で用いられるプロセスについて句分割テンプレートとコンポーネント変換用のテンプレートを事前に作成した．

本実験では複数クエリ統合による実行時間の削減効果のみを評価対象としているため，本実験を実施するのに十分な，現時点で一般的な性能の PC (Intel Core i5 2.67GHz, 4GB メモリ) 1 台を用いて測定を行っている．本実験環境についての性能に対する実行時間の多寡については評価を省略する．

### 4.1 複数クエリ統合の処理型毎の実行時間

各処理型について，複数クエリ統合を行わない場合，句分割による統合と入れ子による統合の二段階の統合を行った場合にそれぞれ生成されたコンポーネントの実行時間を測定した．実行時間は，同じデータセットについて測定を 3 回行ったものの平均とした．実行時間の測定結果を**図 6** に示す．図 6 中の (a) はストリーム型の処理を Esper で実行した際の実行時間，(b) は蓄積型の処理を Hive で実行した際の実行時間である．また，それぞれの件数における複数クエリ統合を行うことで削減される実行時間の割合を平均したものを実行時間削減率として**表 2** に示す．

**表 2** 実行時間削減率

**Table 2** Quotients of the reduction of execution time

| | ストリーム型 | 蓄積型 |
|---|---|---|
| 句分割+入れ子 | 0.600 | 0.108 |
| 入れ子のみ | 0.345 | 0.299 |

句分割による統合によって，「中原駅降車で絞込み」，「日時を時刻に変換」，「男性に絞込み」の 3 個のプロセスの統合クエリ A と「18〜20 時に絞込み」，「ユーザ ID のみ抽出」の 2 個のプロセスの統合クエリ B が生成され，入れ子による統合によって，統合クエリ A を統合クエリ B のサブクエリとして 1 つのクエリに統合される．

句分割による統合と入れ子による統合の二段階の統合を実施することで，Esper では実行時間が 60 パーセント削減される一方で，Hive では 10 パーセント程度の削減に留まることが確認された．

### 4.2 句分割による統合の実施による実行時間への影響

句分割による統合の実施の有無による実行時間への影響を確認するため，入れ子による統合のみを実施した場合に生成されたコンポーネントの実行時間を測定した．前節と同様，実行時間は同じデータセットについて測定を 3 回行ったものの平均とした．なお，測定に用いたイベントデータは前節での測定時と異なるため，複数クエリ統合を行わない場合の実行時間も測定した．実行時間の測定結果を**図 7** に示す．図 7 中の (a) はストリーム型の処理を Esper で実行した際の実行時間，(b) は蓄積型の処理を Hive で実行した際の実行時間である．また，前節と同様に実行時間削減率を算出して表 2 に示す．

入れ子による統合によって，「中原駅降車で絞込み」から「ユーザ ID のみ抽出」までのプロセスが全て統合され，4 段の入れ子を持った統合クエリが生成される．

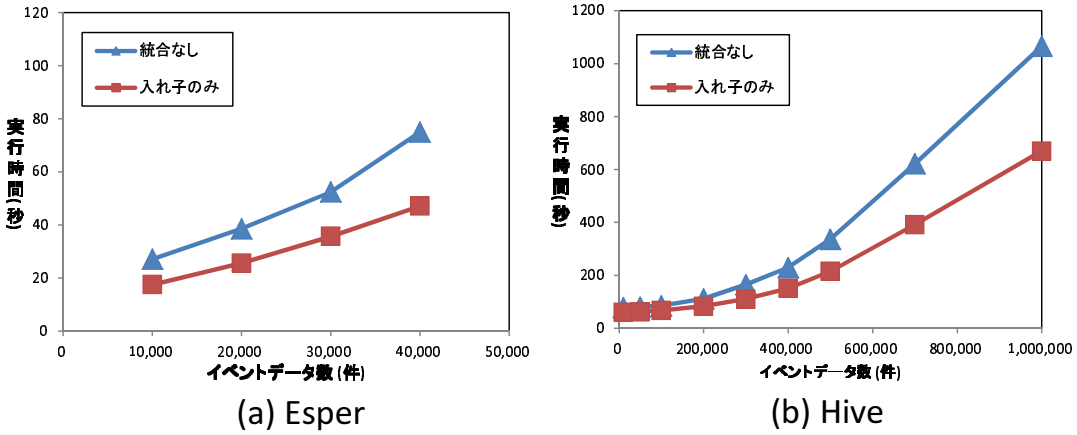


図 7　入れ子による統合のみを行った場合の実行時間

**Fig. 7**　Execution time of the components integrated by nesting

句分割による統合を行わない場合，Esper では実行時間削減率が 35 パーセント程度に低下するのに対し，Hive では実行時間削減率が 30 パーセント程度に改善されることが確認された．

### 4.3　議論

Esper によるストリーム型のプロセスの場合，コンポーネント間のデータの送受信はキューを介して行われる．図 5 の DFD は，句分割による統合の実施有無に関係なく，「中原駅降車で絞込み」から「ユーザ ID のみ抽出」までのプロセスを統合したコンポーネントに変換され，キューは統合前の 6 個から「鉄道イベント」，「対象ユーザ」の 2 個に削減される．Esper での複数クエリ統合による実行時間削減率が高いことから，コンポーネント間のキューによる遅延が大きく，キューの数を減らすことによる効果が高いことがいえる．

ストリーム型において入れ子による統合を実施する際，`INSERT INTO` 句によって前段のプロセスの処理結果を内部ストリーム化し，一時的に PC のメモリ上に保存する．句分割による統合を行わない場合に実行時間削減率が低下するのは，内部ストリームによるクエリ間のイベントデータの受け渡し回数が多くなることが原因だといえる．

Hive による蓄積型のプロセスの場合，コンポーネント間のデータの送受信は HDFS 上のファイルを介して行われる．HiveQL で記述された処理は，Hive によって多段の Map 処理と Reduce 処理に変換され実行される [13]．Map 処理や Reduce 処理の各処理間にて中間データが生成され一時的に HDFS に保存される．Hive での複数クエリ統合による実行時間削減率が Esper よりも低いのは，統合クエリを Hive で処理する際にも統合前のクエリと同様に中間データが生成されるため，コンポーネント数の削減効果が低いからであるといえる．

一方で，蓄積型において句分割による統合を行わず入れ子による統合のみ実施した場合，句分割による統合を行った場合よりも実行時間削減率は改善された．これは，Hive 内部で行われるクエリ最適化が複雑なクエリよりも多段の入れ子を持ったクエリの方がより効率的に行われるのが原因であると予想される．

このように，Esper と Hive では句分割による統合の実施による影響が異なっている．実行時間削減率を考慮すれば，ストリーム型のプロセスは句分割による統合と入れ子による統合の両方の実施，蓄積型のプロセスは句分割による統合は行わず入れ子による統合のみの実施が適しているといえる．

## 5.　関連研究

クエリ最適化 [8] は，データベース管理システム等の処理エンジン上でのクエリ実行計画を最適化し高速に実行する技術であり，複数クエリ最適化 [9], [12] は，処理エンジンにおいて同時に到着した複数のクエリの実行計画をまとめて最適化する技術である．本稿の複数クエリ統合は，逐次的に処理されるクエリをコンポーネント生成時に統合する技術であり，処理エンジン上で実施されるクエリ最適化や複数クエリ最適化と組合せて使用することが出来る．

処理エンジン外でクエリを変換する手法として，Yang ら [14] は，元の関係で記述された複数の句からなるクエリを，既に演算した関係から演算出来るように変換する手法を提案している．また，Ahmed ら [5] は，処理エンジン上での実行時間削減のためにサブクエリの展開などを行う手法を提案している．しかし，コンポーネント数を削減するために複数のクエリを統合する技術はまだ無い．

開発実行環境でクエリ作成・実行を支援する手法として，Peterson ら [11] は，クエリを構成する構成要素をテンプレート化して，それをエディタ上で組合せてクエリを構成する手法を提案している．また，Ahmed ら [4] は，クエリをブロックに分割して，それぞれの結果を他のクエリで再

利用する手法を提案している．CEP の開発環境としては，市販されている様々な製品がデータフローの記述によってクエリを作成出来るようになっている．StreamBase[2] や Sybase[3]，Oracle[1] の製品では，選択，射影，結合等のプリミティブな演算をフローで繋げてクエリを編集する．さらに Oracle の製品は，作成したクエリをフローで繋げてアプリケーションを作成するエディタも備えている．しかし，これらの手法によって作成したクエリを個々のコンポーネントで処理する場合，データ送受信のオーバーヘッドが大きいという問題があり，その問題の解決方法についてはいずれの手法でも言及されていない．

## 6. おわりに

本稿では，DFD から変換されるコンポーネント間のデータ送受信のオーバーヘッドを削減することを目的として，複数クエリ統合を実施することで少数のコンポーネントに変換する手法を提案した．複数クエリ統合は，蓄積型およびストリーム型のクエリ処理エンジンによって実現されるプロセスを対象として，DFD 中のプロセスの順序の維持の有無に応じて，句分割による統合と入れ子による統合の手法を組み合わせて段階的に行うことで，サブクエリ数の少ない入れ子形式の統合クエリを生成することが可能である．

仮想事例の DFD による実験では，句分割による統合，入れ子による統合の二段階のクエリ統合を実施することで Esper では実行時間を 60 パーセント削減出来る一方，Hive では 10 パーセントの削減に留まることが確認された．しかし，句分割による統合を行わず入れ子による統合のみを実施した場合，Hive での実行時間削減率は 30 パーセントに改善した．これらの結果より，処理エンジンが Esper の場合は句分割による統合を実施すべきであり，Hive の場合は実施すべきではないことがわかった．

句分割による統合は，入れ子による統合の際のサブクエリを削減するという意味を持っているが，処理エンジンによってサブクエリの扱い方が異なるため，使用する処理エンジンに応じて句分割による統合を行うか否かを適宜判断する必要がある．Hive のクエリ最適化アルゴリズムは，句分割による統合との相性があまり良くないと予想されるが，本稿ではその検証までは行っていない．従って，本手法による複数クエリ統合が各処理エンジンにおけるクエリ最適化にもたらす影響の調査，ならびにクエリ最適化と相性をより良くするための複数クエリ統合手法の改良が今後の課題として挙げられる．

### 参考文献


[1] : Oracle Complex Event Processing, Oracle Corporation (online), available from ⟨http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html⟩ (accessed 2012-08).

[2] : StreamBase CEP, StreamBase Systems, Inc. (online), available from ⟨http://www.streambase.com/products/streambasecep/⟩ (accessed 2012-08).

[3] : Sybase Aleri Event Stream Processor, SYBASE, Inc (online), available from ⟨http://www.sybase.jp/products/financialservicessolutions/complex-event-processing⟩ (accessed 2012-08).

[4] Ahmed, R.: Reusing optimized query blocks in query processing (2007). US Patent 7,246,108.

[5] Ahmed, R., Lee, A., Witkowski, A., Das, D., Su, H., Zait, M. and Cruanes, T.: Cost-based query transformation in Oracle, *Proceedings of the 32nd international conference on Very large data bases*, VLDB Endowment, pp. 1026–1036 (2006).

[6] Dean, J. and Ghemawat, S.: MapReduce: Simplified data processing on large clusters, *Communications of the ACM*, Vol. 51, No. 1, pp. 107–113 (2008).

[7] DeMarco, T.: *Structured Analysis and System Specification*, Prentice Hall PTR, Upper Saddle River, NJ, USA (1979).

[8] Ioannidis, Y.: Query optimization, *ACM Computing Surveys (CSUR)*, Vol. 28, No. 1, pp. 121–123 (1996).

[9] Kalnis, P. and Papadias, D.: Multi-query optimization for on-line analytical processing, *Information Systems*, Vol. 28, No. 5, pp. 457–473 (2003).

[10] Olston, C., Reed, B., Srivastava, U., Kumar, R. and Tomkins, A.: Pig latin: a not-so-foreign language for data processing, *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ACM, pp. 1099–1110 (2008).

[11] Peterson, T.: QUERY TEMPLATES WITH FUNCTIONAL TEMPLATE BLOCKS (2008). US Patent App. 12/131,263.

[12] Sellis, T.: Multiple-query optimization, *ACM Transactions on Database Systems (TODS)*, Vol. 13, No. 1, pp. 23–52 (1988).

[13] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P. and Murthy, R.: Hive: a warehousing solution over a map-reduce framework, *Proc. VLDB Endow.*, Vol. 2, No. 2, pp. 1626–1629 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1687553.1687609⟩ (2009).

[14] Yang, H. and Larson, P.: Query transformation for PSJ-queries, *Proceedings of the 13th International VLDB Conference*, pp. 245–254 (1987).

[15] 野村佳秀，木村功作，栗原英俊，山本里枝子，山本晃治，徳本　晋：DFD を用いた大規模イベントデータの分析支援環境，ソフトウェアエンジニアリングシンポジウム 2011 (2011).