自律分散型データ共有更新システムにおける 共有ポリシー実装のためのアーキテクチャの検討

沖野　雄哉† 　若林　勇弥†† 　清水　敏之† 　加藤　弘之††† 　吉川　正俊†

† 京都大学大学院情報学研究科　〒 606–8501 京都府京都市左京区吉田本町
†† 京都大学工学部情報学科　〒 606–8501 京都府京都市左京区吉田本町
††† 国立情報学研究所アーキテクチャ科学研究系　〒 101–8430 東京都千代田区一ツ橋 2-1-2
E-mail: †okino@db.soc.i.kyoto-u.ac.jp, ††wakabayashi@db.soc.i.kyoto-u.ac.jp,
†{tshimizu, yoshikawa}@i.kyoto-u.ac.jp, †††kato@nii.ac.jp

あらまし　現在，複数のデータベース間でデータ共有を行う場面が増加している．データ共有を行うことによって，単一のピアで発見できなかったデータの不整合や誤りに気づくことがある．このことから，共有先のピアによるデータの更新をオリジナルのピアへ反映させることが要求されている．一方で，各ピアは意図しないデータ共有を行わないようにするため，データの輸出入に関する独自のルール（以下共有ポリシーと呼ぶ）を設定し，フィルタリングできることが望ましい．本論では，双方向変換の技術を応用して各ピアの設定した共有ポリシーに従い自律的にデータを共有するシステムである SKY アーキテクチャを概観し，共有ポリシーを反映した共有の実現に向けて考察する．

キーワード　データ共有，双方向変換，P2P

1　はじめに

現在，膨大なデータが分散的に各所で生成され，単一の管理主体のみでデータの管理を行うことが難しくなって来ている．類似するデータであったとしても，複数の管理主体 (以降ピアと呼ぶ) で独立してデータを管理していることも多く，さらにそれぞれのピアが同じ形式でデータを保持・公開しているとは限らない．そのため利用者は様々な形式に対応した処理をデータ管理者ごとに記述する必要がある．このことから，管理主体のデータ保持形式に依存せずに，複数ピアによって共同管理が行えるシステム [1] が要求されている．

さらにこのようなデータ共有更新システムを利用する際には，関連するピアの全容が把握できない場合があり，局所的な信頼関係に基づいたフィルタリングが不可欠である．各ピアは輸出入するデータのフィルタリングに関するルール (以下共有ポリシーと呼ぶ) を独立して自由に設定できることが望ましい [2].

本研究では，文献 [3] で提案されているデータ共有の仕組みを SKY アーキテクチャと呼び，SKY アーキテクチャにおける共有ポリシーの実装について議論する．SKY アーキテクチャでは，各ピアが記述した共有ポリシーに従って自律分散的にデータ共有更新を行うことができる．このシステムでは共有リポジトリ SR(Shared Repository) 上で provenance 情報を付与した共有データを集約する．さらに各ピアに対して制御テーブル CT(Control Table) を導入し，データの輸出入に関するフィルタリングを行う．各ピアの基テーブル BT(Base Table) と CT 間，CT と SR 間の同期には双方向変換を応用し，共有ポリシーについては Datalog を記述することで意図したデータ共有を自律的に行うことを実現している．輸入に関するポリシーを IP(Import Policy)，輸出に関するポリシー EP(Export Policy) と呼び，共有に参加するピアはこれらを独立に設定することが可能である．

このシステムの応用例として，研究者が論文にキーワードを付し，さらにそれを他の研究者と共有することで，論文のキーワードを共有管理するという状況が考えられる．各研究者は，元々他の研究者とは独立してデータを管理しており，論文に対して任意のキーワードを付している．この状況下で提案システムを応用し，共有ポリシーに従い自分のキーワード付与を他の研究者へ公開したり，他の研究者のキーワード付与を輸入することでそれぞれが自己最適化したデータベースを構築するということが可能となる．さらにキーワード共有管理で考えられる共有ポリシーの具体例を示し，それらの Datalog による記述方法についても具体的に示す.

またこの SKY アーキテクチャの実装を目指す上で，CT の実装方法が課題となる．双方向変換を用いる際に IP と EP から得られるビューの値域を一致させる必要があるが、値域が一致するようにシステムの利用者が共有ポリシーを記述することは容易ではない．そのため，システム設計を工夫し，この値域の一致を利用者から隠蔽する必要がある．この課題に対し，補助リレーションを導入し，IP から導出される get と EP から導出される get を全域化することにより，利用者が意識することなく前述の課題を解決するアーキテクチャを提案する．

以降の本論文の構成は以下である．2 節で提案システムの概要を説明し，3 節で提案システムの適用を例を用いて説明する．4 節で実装面から明らかになった SKY アーキテクチャの課題を説明し，それに対する提案手法を述べる．5 節にて，SKY アーキテクチャの要求を整理し，他のシステム設計について議論する．6 節で関連研究について紹介し，7 節で本研究をまとめる．

2 SKY アーキテクチャ

2.1 概　　観

まず，SKY アーキテクチャ [3] について基本的な構造について説明する．このシステムは，各ピアが元々保有していた基テーブル BT(Base Table) に対して，制御テーブル CT(Control Table) および共有テーブル SR(Shared Repository) を導入する．BT から SR へのデータの更新反映，SR から BT へのデータの更新反映をそれぞれ輸出，輸入と呼び，BT と CT 間の双方向変換によって輸出に関する制御を行い，SR と CT 間の双方向変換によって輸入に関する制御を行う．輸入に関するポリシー IP(Import Policy) と輸出に関するポリシー EP(Export Policy) は，それぞれ独立した Datelog で記述される．

システムの全体像としては図 1 のように表される．図 1 では共有に参加しているピア数は 2 であるが，任意のピアが共有に参加することが可能である．また，共有に参加するピアには，それぞれ一意の ID が割り当てられ，以降 pid と表記する．

2.2 スキーマ

各関係のスキーマについて述べる．BT のスキーマは，BT(p, id，v) と表される．p が provenance を保持する属性，id が関係 BT における主キー，v がその他の値を保持する属性である．p は元々どのピアで作成され，どのピアによって更新されたかを表す属性であり，更新履歴を参照したフィルタリングを可能にするための補助的な属性である．詳細な構造については次節にて述べる．v は，他の補助的な属性と異なり本来共有したい情報を保持する属性である．CT のスキーマは BT と同様の属性を持ち，CT(p，id，v) と表される．SR のスキーマは SR(p, acc，id，v) と表され，BT，CT の持つ属性に加えて，acc という属性を持つ．この属性 acc は，このタプルがどのピアによって輸入されたかを表し，輸入したピアの pid の集合として値を持つ．例えばある輸出されたタプルが，3 つのピアに輸入され，その pid が P1，P2，P3 であれば acc の値は {P1，P2，P3} と表現される．この属性を持つことで，ピアは他のピアの輸入状況を鑑みてタプルを輸入するか決定することが可能となる．

3 共有ポリシー

各ピアの持つ共有ポリシーは，輸入に関するポリシーを記述した IP(Import Policy) と，輸出に関するポリシーを記述した EP(Export Policy) の 2 つに分けられる．これらのポリシーは，BIRDS [4] を応用し，Datalog を記述することで実現する．BIRDS [4] を用いることによって，Datalog で記述した putback-based な put の表現から，get を一意に導出し，さらに SQL で get に対応したビュー定義と put に対応したトリガー定義を出力することができる．本研究では，BIRDS を利用しているため，本来共有ポリシーには put にあたる Datalog を記述すべきであるが，簡単のため以降の例では get として記述する．

3.1 provenance

SKY アーキテクチャでは，ピアの更新履歴に基づいてタプルのフィルタリングが可能である．これを可能にするため，各テーブルは属性 p を持っており，どのピアによって作成，更新されたかを保持する．共有ポリシーに p やそれらに関わる関数を用い，選択を記述することで provenance を参照した記述を行うことができる．

以降属性 p の構造とそれに関わる関数について述べる．まず，p は次のように再帰的に定義される．

$$p := tid@pid \tag{1}$$
$$| p + p \tag{2}$$
$$| p * p \tag{3}$$
$$| p@pid \tag{4}$$

(1) がこれ以上分割できない最小の構造であり，tid と pid のみで表現される．このシステムに新しいタプルが作成された時 p はこの構造をとる．異なる BT において同じタプルが作成された時には (2) によって定義され，複数の BT を結合して生成されたタプルについては (3) のように*を用いて表現される．また，更新履歴については，(4) の@によって更新を行ったピアの pid を append することで表現する．具体的には，ピア X で作成されたものがピア Y で更新されたタプルの p は，tid@X@Y と表される．さらに，ポリシー定義時の利便性のため，この構造を持つ p に対して，関数 ulog と org を定義する．

$$ulog(p) = \begin{cases} [pid] & \text{if p == tid@pid} \\ [ulog(p1) \cup ulog(p2)] & \text{if p == p1 + p2} \\ [ulog(p1) \cup ulog(p2)] & \text{if p == p1 * p2} \\ (ulog(p1)) + +[pid1] & \text{if p == p1@pid1} \end{cases}$$

$$org(p) = \begin{cases} \{pid\} & \text{if p == tid@pid} \\ (org(p1)) \cup (org(p2)) & \text{if p == p1 + p2} \\ (org(p1)) \cup (org(p2)) & \text{if p == p1 * p2} \\ org(p1) & \text{if p == p1@pid1} \end{cases}$$

ulog によって p の更新履歴の系列，org によって元々所有していたピアの pid の集合が取得できる．例えば，length(ulog(p)) == 1 である時，そのタプルは誰にも更新されていないタプルであるということを意味する．このように，p を定義することで柔軟で利便性の高いポリシーの記述が可能となっている．

3.2 例

本研究の適用例として，「各研究者が論文にキーワードを付す」という状況があげられる．この例について，どのようにポリシーを表現することができるか説明する．まず，3 人の研究者 X，Y，Z が存在するとし，それぞれが元々表 3.2 の情報を BT として保持しているとする．

さらに，各ピアは他のピアと付与したキーワードを共有を行いたいが，ポリシーとして表 2 のような方針でデータ共有を行

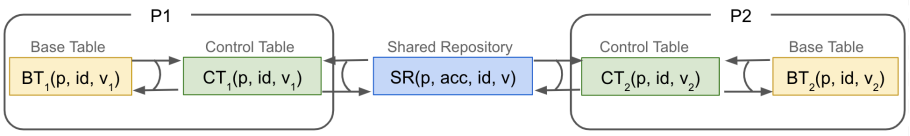

図 1　SKY アーキテクチャ全体像

表 1　各ピアの BT

BT_X

id	paper	keyword
t1	A	データベース
t2	B	情報検索

BT_Y

id	paper	keyword	share
t3	A	情報図書館	true
t4	C	データマイニング	false

BT_Z

id	paper	keyword
t5	B	最適化

表 2　各ピアの共有方針

ピア	方針
X	自身の持つ全てのタプルを輸出し，他の人が公開したタプルを全て輸入したい
Y	自身の持つタプルのうち ‘share’ の値が true であるタプルのみを輸出し，Z が生成したタプルのみ輸入したい
Z	自身の持つ全てのタプルを輸出するが，どのタプルも輸入しない

うと想定する．

各ピアはそれぞれ独立したポリシーを持つ．ピア X は全てのデータを共有によって管理したいが，ピア Y，Z のように部分的に共有を利用したい場合も考え得る．ピア Y はある条件に適合したタプルのみを輸出するが，輸入に関しては特定ピアへの信頼関係からタプルのフィルタリングを行う．ピア Z では，他のピアが生成したタプルを輸入しないが，自身のタプルは全て公開したいというような要求も考えられる．

このように各ピアが共有に独自のポリシーを反映することで，それぞれのピア内で自己最適化したデータベース管理が可能となる．

3.3　共有ポリシーの記述

本節では，前節で説明した表 2 の方針を持つピアに対し，Datalog によるポリシーの記述例を説明する．

まず，EP_X については，以下のように記述することができる．

$$CT_X(p, id, paper, keyword) := BT_X(p, id, paper, keyword)$$

ピア X は自身の持つ全てのタプルを輸出したいので，BT に存在するタプルは全て CT に同期する．一方 IP_X については，以下のように記述することができる．

図 2　SKY アーキテクチャにおける CT の実装

$$CT_X(p, id, paper, keyword) :=$$
$$SR(p, acc, id, paper, keyword)$$

EP_X と同様全てのタプルを輸入すれば良い．続いて，EP_Y については，自身の持つタプルにのうち ‘share’ が true であるもののみ輸出するので，

$$CT_Y(p, id, paper, keyword) :=$$
$$BT_Y(p, id, paper, keyword, share), share = true$$

のように， share 属性による選択を行えば良い．IP_Y については次のように表現できる．

$$CT_Y(p, id, paper, keyword) :=$$
$$SR(p, acc, id, paper, keyword), Z \in org(p)$$

org を用いた選択を行うことによって，Z がオリジナルの作成者であるタプルのみ輸入することができる．続いて，EP_Z については，自身の持つ全てのタプルを輸出するので，X と同様の定義で十分である．IP_Z については，例えば，自分自身が輸出したもののうち他の人によって更新されていないものを輸入するというポリシーの記述が考えられる．

$$CT_Y(p, id, paper, keyword) :=$$
$$SR(p, acc, id, paper, keyword), Z \in org(p),$$
$$len(ulog(p)) == 1.$$

このように 2 節にて提案した SKY アーキテクチャを利用することで，柔軟性に富んだ様々なポリシーを表現し，共有に参加するピアの意図を反映したデータの共有を行うことができる．

4　提 案 手 法

4.1　SKY アーキテクチャの課題

SKY アーキテクチャでは CT が，BT と CT 間の双方向変換と，CT 間と SR 間の双方向変換と，二つの双方変換から定義される．このような cospan の関係にあるビューを実装するためには，図 2 のように二つの双方向変換からそれぞれビューを生成し，それらを同期する実装が考えられる．

しかしながら，このシステムにおいて輸出と輸入のポリシーは独立した Datalog で記述されることから，必ずしも IP から導出される get と EP から導出される get の値域が一致す

るとは限らない．具体例として，表 2 における Y の共有ポリシーについて考える．ピア Y のポリシーに基づいて，CT_{BT} と CT_{SR} にどのようなタプルが入り得るかを考えると，IP_Y から，CT_{SR} は $share = true$ を満たすタプルのみ受け付け，EP_Y から，CT_{BT} は $Z \in org(p)$ を満たすタプルのみ受け付ける．ここで SR にタプル $(p, acc, id, paper, keyword, true) = \{'tid@G'$, $acc1, id1, p1, k1, false\}$ が存在する場合について考える．このタプルは，IP_Y と合致しており，CT_{SR} に輸入される．最終的にこのタプルをピア X の BT まで輸入してきたいが，CT_{BT} の値域外のタプルであるため，CT_{BT} 経由で BT まで同期することができない．図 2 のアーキテクチャでは，ポリシーの書き方によって意図した更新同期と異なる挙動となる可能性がある．

加えてシステムの利用者がこの値域の一致を意識しながら共有ポリシーを記述することは容易ではない．これらのことから SKY アーキテクチャの利用者が，この値域の一致について意識せずに，IP と EP を独立に記述できるようなシステム設計にする必要がある．

4.2 補助テーブルの導入

前節で述べた，値域の一致をアーキテクチャレベルで実現する手法として put の全域関数化を説明する．

putback-based な双方向変換技術を用いる際に，高々一つの補助テーブルを導入することによって put から導出される get の値域を全域化する手法が提案されている [5]．この全域化された双方向変換は元の双方向変換から決定的に導出可能である．

この手法を取り入れることで，IP から導出される get と EP から導出される get をどちらも全域化することができ，全域化された二つのビューを同期することで，cospan の関係にあるビューである CT を関係データベース上に再現することが可能である．

全域関数化を用いて補助テーブル AT(Auxiliary Table) を取り入れたアーキテクチャが図 3 である．このアーキテクチャでは，IP または EP に当てはまらないタプルは AT へと同期される．例えば，4.1 節で挙げたタプル $\{'tid@G', acc1, id1, p1$, $k1, false\}$ について同期の流れを考えると，まず SR から CT_{SR}^{total} へと同期され，続いて CT_{BT}^{total} へと同期される．ここで，CT_{SR}^{total} と CT_{BT}^{total} は全域化されているため，どのようなタプルでも同期することが可能である．さらにこのタプルは IP に合致しないため，CT_{BT}^{total} から AT_{BT} へと最終的に同期される．

IP または EP に合致しないタプルは AT へと同期されることになるため，AT と BT を合わせたものとして利用者に提供するための補助的な仕組みがさらに必要となると考えられる．複雑なアーキテクチャとなり，トランザクションの管理や処理速度が課題となると考えられる．

5 議　　論

SKY アーキテクチャでは，putback-based な双方向変換を用いたポリシーを記述することで，フィルタリングと更新同期

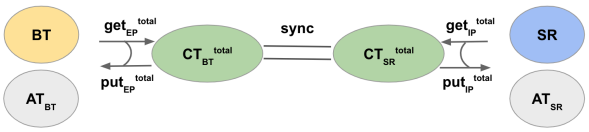

図 3　SKY アーキテクチャ 全域関数化

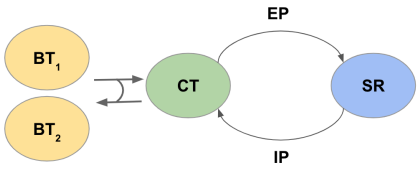

図 4　輸出入の分離を行ったアーキテクチャ

を同時に実現することができたが，ポリシーの記述という観点からはユーザが意図したポリシーを putdelta の Datalog で直感的に表現することが難しいという課題も存在する．

ここで CT と SR の同期処理に着目した時に，必ずしも双方向変換技術を応用する必要はない．CT へ変更があった場合，SR にその変更を波及させ，SR に変更があった場合，CT へその変更を波及させれば十分にシステムの要件を満たすことができる．これらを踏まえ，ピアごとに異なるスキーマの差異の緩衝にのみ双方向変換を用い，SR と CT の更新同期については別の技術を用いる図 4 のアーキテクチャについても検討の余地があると考えている．

このアーキテクチャでは輸出入に関するポリシーは，実質的に欲しいデータを選択・射影するクエリを書けば良いだけであり，putdelta に当たる Datalog を書く必要はない．より直感的にポリシーを記述することができる．加えて双方向変換で CT と SR の更新同期をとる場合，変更が同期的に波及していくが，この提案手法では，必要なタイミングで非同期に更新を同期することも可能であり，より柔軟に同期処理を制御することが可能となる．

6 関 連 研 究

6.1 協調型データ共有システム

複数のスキーマが異なるピア間でデータを共有する手法に，Collaborative Data Sharing System [6] がある．この手法では，schema mappings によってどのピアからデータを取得するか記述し，local rejection と local contributions を定義し，共有するデータを制御している．この手法では，データ共有に参加するピアは基本的に二者間でのデータ共有に着目しているが，本研究の提案手法では，provenance を各タプルが保持し，それによってピアが更新履歴を参照した細かい制御を行える．

6.2 双方向変換

本研究では，BT と CT 間，CT と SR 間のデータの同期を双方向変換の技術を応用した．双方向変換は，順方向変換 get と逆方向変換 put のペアから構成される [7]．

順方向変換 $V = get(S)$ は，ビュー定義であり，ソースとな

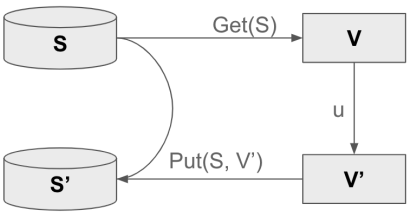

図 5　双方向変換

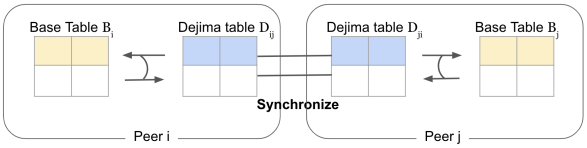

図 6　Dejima アーキテクチャ

る関係から，ターゲットであるビューを導出する．逆方向変換 $S' = put(S, V')$ はビューへの更新をソースへ反映するもので，一般的に get は単写とは限らず全てのソースデータがターゲットデータに反映するとは，限らないため，引数にソースと更新後のビューを与え，更新を反映したソースを導出する．この双方向変換を利用することで，ソースとビューの更新同期を取ることができる．また，双方向変換での整合性を保証するための主な性質に GetPut と PutGet があり，次の round-tripping law を満たすことが求められる．

$$put(S, get(S)) = S \quad \text{(GetPut)}$$
$$get(put(S, V')) = V' \quad \text{(GetPut)}$$

さらにこの GetPut と PutGet の性質を満たした双方向変換を導出する手法に，putback-based の双方向変換技術が提案されている [8] [9]．これらの中には，関係データベース上で更新可能なビューを導出することができる putback-based の双方向化技術 BIRDS [4] が提案されており，LVGN-Datalog で putdelta に当たるプログラムを記述することで，双方向変換を関係データベース上の関係とビューで表現することが可能である．本研究では，この手法を利用する．

6.3　Dejima アーキテクャ

Dejima アーキテクチャ [10] は双方向変換技術を応用し，二つの関係間でデータ共有・更新同期を行うシステムである．基テーブル BT 上にあるデータのうち，他方のピアに輸出したいものを Dejima テーブルと呼ばれるビューへ反映させ，ペアの Dejima テーブル同士の同期を取ることで双方向のデータ同期を実現する．

本研究で提案した SKY アーキテクチャでは，自分が輸出したタプルが他者によって更新された場合，その更新ををを受け入れない，自分のタプルは上書きさせないというポリシーを書けるが，一方 Dejima アーキテクチャにおいて輸出したタプルが変更された時，その変更は自動的に反映される．ピア間の信頼に基づいて同期を取るアーキテクチャだと言える．

7　おわりに

本論文では，双方向変換技術を応用し，共有ポリシーを反映した自律分散的なデータ共有・更新が行えるシステムである SKY アーキテクチャについて概観し，導入した補助的な属性や関数を利用し，Datalog での柔軟なポリシー記述が行えることを，例を用いて説明した．さらに，SKY アーキテクチャの CT の実装には値域の不一致という課題が存在し，補助テーブルを導入し CT を全域関数化することで，利用者が意識せずとも前述の課題を解決することができるアーキテクチャを提案した．

謝辞　本研究の一部は JSPS 科研費 JP17H06099, JP18H04093，JP18K11315 の助成を受けたものです．

文　献

[1] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[2] Grigoris Karvounarakis, Todd J. Green, Zachary G. Ives, and Val Tannen. Collaborative data sharing via update exchange and provenance. *ACM Trans. Database Syst.*, 38(3):19:1–19:42, 2013.

[3] 清水 敏之, 加藤 弘之, and 吉川 正俊. 『科学メタデータクリーニングのための自律分散型データ共有・更新システムに向けて』. *DEIM Forum 2020*, D8-2.

[4] Van-Dang Tran, Hiroyuki Kato, and Zhenjiang Hu. Programmable view update strategies on relations. *PVLDB*, 13(5):726–739, 2020.

[5] 田中 順平, Van-Dang Tran, 加藤 弘之, and 胡 振江. 『双方向化技術によるデータベーススキーマの共存』. *DEIM Forum 2020*, H6-4.

[6] Grigoris Karvounarakis, Todd J. Green, Zachary G. Ives, and Va l Tannen. Collaborative data sharing via update exchange and provenance. *ACM Transactions on Database Systems*, 38(3):19:1–19:42, 2013.

[7] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3), May 2007.

[8] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu. Bigul: a formally verified core language for putback-based bidirectional programming. In *Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, pages 61–72, 2016.

[9] Sebastian Fischer, Zhenjiang Hu, and Hugo Pacheco. The essence of bidirectional programming. *Science China Information Sciences*, 58(5):1–21, 2015.

[10] Y. Ishihara, H. Kato, K. Nakano, M. Onizuka, and Y. Sasaki. Toward bx-based architecture for controlling and sharing distributed data. In *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 1–5, 2019.