DEIM Forum 2019 D4-4

データストリーム管理システムに関する再考

杉浦　健人†　　石川　佳治†

† 名古屋大学大学院情報学研究科　〒 464–8603 愛知県名古屋市千種区不老町
E-mail: sugiura@db.is.i.nagoya-u.ac.jp, ishikawa@i.nagoya-u.ac.jp

あらまし　リアルタイムに生成されるデータを活用したいという要求は多く，近年は高スループットな処理能力を持つストリーム処理システムが商用及びオープンソースソフトウェアで活発に開発されている．一方，そうしたストリーム処理システムは一貫性などの ACID 特性を保証をしていないものが多く，それらの保証はアプリケーション開発者の実装に委ねられる．しかし，データベース分野でトランザクション処理が発展し続けているように，ACID 特性を満たすデータ処理への要求は根強い．そこで，本稿ではデータストリーム管理システムに再度着目し，データモデルやトランザクション処理について調査，検討する．特に，イベントタイムに基づく処理順を考慮した処理について，既存研究のアプローチを基に議論する．

キーワード　データストリーム管理システム，データストリーム処理，トランザクション処理

1 はじめに

データの大規模化に伴い，大量のデータをリアルタイムに処理するためのストリーム処理システムが商用ソフトウェア・OSS（open source software）を問わず活発に開発されている．インターネットを始めとする情報技術の発展によりビッグデータ時代とも呼ばれる時代が訪れ，データの大量生成と大量消費が行われている．2000 年代には MapReduce [9] に基づく Hadoop [3] や Spark [4] など，主に静的な分散処理システムによってそうしたビッグデータを処理していた．しかし，それらのシステムはバッチ処理に基づいており，日々加速する意思決定のリアルタイムな支援には不足があった．そうしたリアルタイムなデータ処理の要求に応えるために，最近 10 年では Flink [2] や Spark Streaming [6]，Samza [5] を始めとする分散ストリーム処理システムが活発に研究，開発されており，そのユーザ数を増やしている．

一方，ストリーム処理の多くは**連続的 *OLAP*（*online analysis processing*）のみ**を対象としており，ACID 特性 [13] の保証は暗黙的に行われている．通常，古典的なストリーム処理は以下の 3 つの仮定を置いている．

（1） 全てのストリームは append-only である．つまり，あるタプルに対する更新をタイムスタンプにより区別（元のキーとタイムスタンプの組を主キーとみなす）し，更新履歴を全て保持したテーブルをストリームとして扱う．

（2） 各処理はタイムスタンプの情報を用いることで，自身の処理で必要なデータを読める．

（3） 全ての連続的問合せ処理のデータフローは非循環である．つまり，問合せ処理の流れを DAG（directed acyclic graph）で表せる．

このような前提を置く場合，ACID 特性のうち isolation は暗黙的に満たされる．全てのストリームが append-only であるということは，あるデータ x に対する書き込み $w(x)$ は一度しか発生しないことを意味する．したがって，トランザクションをどのように同時実行しても書き込みに対して競合は発生せず，あるトランザクションの処理に必要なデータがすでに書き込まれていることさえ保証すればよい．その他，ある DAG で表される連続的問合せを一つのトランザクションとみなす場合，atomicity 及び durability の保証は exactly-once の保証とおおよそ等しく，ストリーム処理システムの多くはこれらの要素から ACID 特性の保証を暗黙的に行っているとみなせる．

しかし，近年ではより複雑なストリーム処理を記述したいという要求から，*stateful* な処理，つまり処理の**内部状態**を考慮した処理を実現しようとする動きがある．これはプログラム中で更新される変数の値をデータベースに格納し，時間的に後の処理もしくは DAG 上で後段に存在する処理でデータベースに格納した値を参照することだと言い換えられる．つまり，上述した「3) 全てのデータフローが非循環である」という仮定が成り立たないことを意味する．

データフローが循環を持つ場合，トランザクションの処理順，言い換えればトランザクションの linearizability の保証が重要となる．簡単な例として，図のような 3 つのトランザクションからなるストリーム処理を考える．なおこれ以降，トランザクション T_x が T_y の後に実行されることを $T_x \prec T_y$ で表す．図において，トランザクション T_a 及び T_b とこれらの書き出した値を読む T_c の間には順序関係 $T_a \prec T_c$ 及び $T_b \prec T_c$ が存在する．一方で，トランザクション T_b 及び T_c は内部状態をデータベース中のテーブルを用いて共有しており，これらの間にも順序関係が存在する．つまり，トランザクション T_b が読む値は，何らかのタイミングで T_c によってテーブルに書き込まれた値であり，その値がいつ書き込まれたかによってこれらの順序関係が $T_b \prec T_c$ なのか $T_c \prec T_b$ なのか定まる．このように連続的問合せとして事前にトランザクションが登録してある場合，各トランザクション間が満たすべき *read-from* の関係から，一部のトランザクション間に事前に順序関係が定まる．

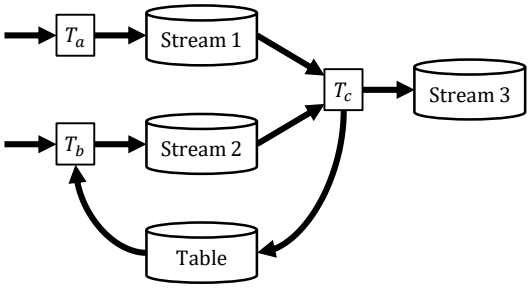

図 1　内部状態を考慮したストリーム処理のデータフロー

そこで本研究では，ストリーム処理におけるトランザクション処理を実現するために，処理の linearizability の保証方法について議論，検討する．つまり，トランザクション処理における ACID 特性の consistency として， linearizability を考慮したデータストリーム管理システム（data stream management system: DSMS）を提案する．ストリーム処理におけるトランザクション処理を提案する既存研究は存在する [7,11] ものの，ユーザが記述する DAG に基づきトランザクションの順序関係を定義しており，本来満たされるべき順序関係を適切に記述できていない．それに対し，本研究ではトランザクション間の read-from 関係に基づき順序関係を定める．更に，トランザクションの生存時間を考慮することで，同時刻に行われる処理の順序関係と異なる時刻に行われる処理の順序関係を分離し，適切にトランザクション間の順序関係を記述する．

本稿の構成は以下の通りである．まず，2 章では関連する研究とその問題点を紹介する．

2　関 連 研 究

データストリーム処理システムにおけるトランザクション処理について検討した既存研究はいくつかある [7,8,11,12]．しかし，Chen らの研究 [8] と Mozafari らの研究 [12] は理論的な面について深く言及していないため，以下では Botan らの研究 [7] と Meehan らの研究 [11] についてのみ紹介する．

Botan ら [7] は，データベース管理システム（database management system: DBMS）におけるトランザクション処理を踏襲しつつ，データストリーム処理のためのトランザクション処理を提案した．Botan らが着目したのは，データストリーム処理システムはデータの入力によって処理が開始する連続的問合せを扱うのに対し，既存の DBMS では基本的に命令（read/write）そのものが入力される one-time 問合せを扱うという点である．つまり，既存の DBMS にはトランザクション間の順序を決定する仕組みが存在しないことを問題として挙げた．そこで，Botan らはまずデータストリームを append-only かつ順序付け可能なタプルの集合として，連続的問合せを one-time 問合せの系列として考え，DBMS と同様のトランザクション処理が行えるものとした．更に，上述のような処理の順序を厳密に定めるために，CSR（conflict serializability）を拡張した CSAO（conflict serializability with arrival ordering）を提案し，データの到着順に合致した処理が行えるシステムを実装した．なお，トランザクション間の順序関係については後述する Meehan らの研究のものがより汎化した定義であるため省略する．

Meehan らの研究 [11] では in-memory DB である H-Store [10] 上でストリーム処理を実装し，トランザクション処理を考慮したデータストリーム処理システムである S-Store を提案した．S-Store では，ストリーム処理の要件として，1 章で仮定として述べた DAG で表されるようなデータフローに基づく処理を挙げている．つまり，トランザクションの順序付けられた実行である．これを表すために，まず S-Store ではあるトランザクション T_i に入力されるストリーム S_i が以下のようにマイクロバッチ [14] に分割されることを前提とした．

$$S_i = \langle s_i.b_1, s_i.b_2, ..., s_i.b_r, ... \rangle \tag{1}$$

更に，マイクロバッチ $S_i = \langle s_i.b_1, s_i.b_2, ... \rangle$ は時刻順に入力されるものとし，ストリーム処理のために満たすべき要件としてマイクロバッチの入力順に合わせたトランザクションの実行を定義した．つまり，あるトランザクション T_i に対するマイクロバッチ $s_i.b_j$ の処理を $T_{i,j}$ で表すとすると，以下の式を満たすことを要件とした．

$$T_{i,1} < T_{i,2} < ... < T_{i,r} < ... \tag{2}$$

更に，データフローにおけるトランザクションの順序を表すため，データフローの DAG をトポロジカルソートすることで得られる以下の式も満たすべき要件とした．なお，T_1 から T_n が同じデータフロー上にあるものとする．

$$T_{1,r} < T_{2,r} < ... < T_{n,r} \tag{3}$$

このような処理を実現するために，S-Store はトリガーによる処理順の制御を実装した．つまり，データストリームをテーブルで表しそれにトリガーを付与することで，上述の順序関係が守られるようシリアルヒストリーを生成している．

しかし，これらの研究では図 1 のような循環のあるデータフローで表される処理の順序関係を適切に表現できない．まず，既存研究ではそもそもデータフローが非循環であることを前提としている．S-Store ではデータベース中のテーブルへのアクセスも考慮しているが，テーブルがデータフローに与える影響は議論しておらず，あるテーブルへの読み書きは一つのトランザクション内で完結していると考えられる．S-Store では階層的なトランザクションも使用可能であるため，あるテーブルにアクセスするトランザクションが複数存在する場合，ユーザによって一階層上のトランザクションとして一つにまとめられることが前提となっていると考えられる．つまり，図 1 であれば T_b と T_c を包含したトランザクションの存在が前提となる．

3　トランザクションの順序関係

3.1　データベース管理システムにおける順序関係

まず，DBMS におけるトランザクション間の順序がどのように決まるかを説明し，DSMS における順序関係を議論するためのアイデアを紹介する．

前提として，DBMS において CSR を考えるとき，全てのトランザクション $\mathcal{T}$ 上の順序関係は前順序で表される．つまり，同じデータ $x \in D$ に対して競合の発生するトランザクション $T_i, T_j \in \mathcal{T}$ の間には下記のように両方向に順序関係が成り立つ．

$$T_i \prec T_j, T_j \prec T_i \tag{4}$$

このうち，いずれかの順序関係を破棄することで $\mathcal{T}$ 上での半順序関係を構築するプロセスが CSR におけるシリアルヒストリーの選択だとみなせる．

つまり，あるシリアルヒストリーが選択されたとき，トランザクションの順序関係は半順序で表せる．例えば，以下のような 5 つのトランザクションの集合 $\mathcal{T}$ を考える．

$$T_a = r(x)w(y), \quad T_b = r(y)w(x), \quad T_c = w(z),$$
$$T_{-\infty} = w(x)w(y)w(z), \quad T_{\infty} = r(x)r(y)r(z)$$

なお，$T_{-\infty}$ 及び T_{∞} は初期状態及び最終状態を確定させるためのトランザクションである．このとき，$\{T_a, T_b, T_{-\infty}, T_{\infty}\}$ は x と y について，$\{T_c, T_{-\infty}, T_{\infty}\}$ は z について競合が発生しており，式 (4) のように $\mathcal{T}$ 上には前順序の関係が成り立つ．次に，以下のシリアルヒストリー h_s を考える．

$$h_s = T_{-\infty}T_aT_bT_cT_{\infty}$$

このとき，$\mathcal{T}$ 上の順序関係は以下の半順序で表せる．

$$T_{-\infty} \prec T_a, \quad T_{-\infty} \prec T_c, \quad T_{-\infty} \prec T_{\infty},$$
$$T_a \prec T_b, \quad T_a \prec T_{\infty}, \quad T_b \prec T_{\infty}, \quad T_c \prec T_{\infty}$$

これらの順序関係は，任意のデータ $x \in D$ に対する処理の競合から生まれる順序である．つまり，シリアルヒストリーにおいて各データに対する read/write 命令の順序が全順序となるために最低限必要な順序関係である．このとき，順序関係の存在しないトランザクションの位置を入れ替えたシリアルヒストリー h'_s は，元のシリアルヒストリー h_s と等価である．例えば，上記において T_c は T_a とも T_b とも順序関係を持たない．したがって，$T_{-\infty} \prec T_c \prec T_{\infty}$ さえ満たしていれば，T_c の位置を変更しても元のシリアルヒストリー h_s と等価な結果が保証される．

このように，DBMS においてトランザクションの順序関係は前順序から始まり，何らかのシリアルヒストリーを選ぶことで半順序へと定まる．しかし，既存研究 [7,11] で指摘されているとおり，ストリーム処理における連続的問合せでは一部のトランザクション間の順序関係が始めから半順序で表される．次節では，この半順序の関係がどのように定まるか述べる．

3.2 データストリーム管理システムにおける順序関係

本節では，ストリーム処理においてトランザクションの順序関係がどのように定まるかを述べる．以下では，まず本研究で扱う時刻の概念について説明し，その後時間に関して拡張したトランザクションを紹介する．最後に，トランザクション間の順序関係の決定方法を述べる．

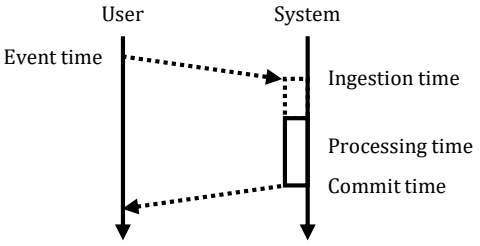

図 2　時刻のイメージ図

3.2.1 ユーザ時刻とシステム時刻

本研究では，以下の 4 つの時刻を考える．なお，各名称は Flink で用いられているものを参考とした [1].

（1） Event time：ユーザが指定する任意の時刻.

（2） Ingestion time：トランザクションがシステムに入力された時刻.

（3） Processing time：トランザクション内の read/write 命令がシステム内で処理された時刻.

（4） Commit time：トランザクションがコミットした時刻.

ある時間軸における各時刻のイメージとしての位置を図 2 に示す．なお，点線の矢印はユーザ・システム間の通信を表し，点線の矩形はシステム内におけるトランザクションの処理の保留を，実線の矩形が実際に処理が行われている時間を表す.

更に，これらの時刻は，event/ingestion time を**ユーザ時刻**，processing/commit time を**システム時刻**として 2 つに分けられる．ユーザ時刻は，言い換えればシステムによって操作できない時刻を表す．Event time はユーザが任意に指定できる時刻であるため，当然システムでは変更できない．Ingestion time も，トランザクションがいつシステムに入力されるかはユーザ次第であり，システム側では操作できない．一方で，システム時刻はシステムが自由に設定できる時刻である．つまり，processing/commit time はそのトランザクションないし命令をいつ処理/コミットするかであり，これらはシステム側で任意に操作できる.

3.2.2 時間を考慮したトランザクション

データストリームを扱う上で重要となるのは，全てのデータが本質的に**時系列データ**となり，連続的問合せにおいて時間に関する条件が本質的に存在する点である．例えば，時間窓を用いた問合せについて考える．窓幅 w のスライディングウィンドウでデータを集約するような連続的問合せを考えたとき，ある時間窓 $[t, t+w]$ の問合せは，暗黙的に時区間 $[t, t+w]$ のデータが全て揃っていることを要求する．つまり，時区間 $[t, t+w]$ に対する read 命令を行う前に，時区間 $[t, t+w]$ への write 命令が処理されていることを前提としている.

これは， CSR において競合の発生するトランザクション間の順序関係のうち，一部はタイムスタンプによって確定することを意味する．例えば，時区間 $[t, t+w]$ のデータ x に対して write を行うトランザクション T_i と時区間 $[t, t+w]$ のデータ x を read するトランザクション T_j が存在するとき，ストリーム処理においてその順序関係は $T_i \prec T_j$ であることが暗黙的に要求される．既存研究 [7,11] ではこの順序関係をユーザが定義す

る処理の DAG やストリームの到着順で表そうとしているが，本質的には上述したようなタイムスタンプに基づく read/write 命令の順序関係である．

なお，データの時系列化はユーザによって明示的に指定されない場合にも行われる．データを読み書きする際に event time に基づいて時刻が指定されている場合，当然それらのデータは時系列データとなる．一方で，データを event time 以外の時刻に基づいて処理する場合もデータは時系列に基づいて処理される．例えば， DBMS は read 命令を processing time で， write 命令を processing/commit time で行うシステムである．つまり，全ての read はその命令を行う時点（ processing time ）で最新の値を読み，全ての write はその命令を行った時刻を基に書き込みかつコミットと同時（ commit time ）に各データの最終状態を確定させる．

以上を基に，本研究におけるストリーム及びトランザクションを定義する．まず，全データとストリームの定義を示す．

［定義 1］ データストリーム管理システムで管理する全てのデータの集合を D で表す．ストリーム S はデータ集合 D を構成する要素であり，以下の性質を持つ．

$$S \subseteq D \tag{5}$$

$$\forall S_i, S_j \subseteq D, S_i \cap S_j = \emptyset \tag{6}$$

また，ストリーム S 中の各データは以下の性質を持つ．ただし，ストリーム中の各データ $d \in S$ はタイムスタンプ，主キー，値の 3 つの組 $d = (t, key, value)$ で表し，それぞれを $d.t$ のような形式で参照する．

$$\forall d \in S, d.t \in \mathbb{N} \tag{7}$$

$$\forall d, d' \in S, d.key = d'.key \wedge d.t = d'.t \rightarrow d.value = d'.value \tag{8}$$

■

以下，議論に必要ない限りあるストリーム S 中のデータ $d \in S$ も単に $d \in D$ で表す．つまり，全てのデータの値をタイムスタンプ及びキーによって一意に指定できるものとして議論する．

次に，データに対する読み書きの命令を以下のように定義する．

［定義 2］ 読み書きの命令を $r(t_{int}, key)$ 及び $w(t, key)$ で表す．ただし，t_{int} はタイムスタンプに関する実区間である．より厳密には，$r(t_{int}, key)$ は以下の式に基づきあるデータ $d \in D$ を読み出す．

$$r(t_{int}, key) := \arg\max_{d \in \{d | d \in D \wedge d.key = key \wedge d.t \in t_{int}\}} d.t \tag{9}$$

また，$w(t, key)$ はデータ $d = (t, key, value)$ を D に追加する命令である． ■

全てのデータアクセスをタイムスタンプに基づいて処理することで，ユーザ時刻に基づく命令を表す．なお，システム時刻に基づく命令は ∞ を用いて表現する．つまり，読み込みは $r([-\infty, \infty), key)$，書き込みは $w(\infty, key)$ として表す．$-\infty$ は過去に行われた全ての書き込みが対象となることを表し，∞ は処理の行われる時間が論理的に決まっていないこと，つまり実行ヒストリーに配置される際にシステム時刻を用いて時刻が割り当てられることを示す．

read/write 命令を定めたことで，命令の集合が満たすべき順序関係を以下のように定められる．

［定義 3］ 命令の集合 op が与えられたとき，以下の式から導かれる op 上の順序関係 $\prec$ をタイムスタンプに基づく順序関係とする．ただし，$W_{t_{int}}(key) = \{w(t, key) | w(t, key) \in op \wedge t \in t_{int}\}$ とする．

$$\forall r(t_{int}, key) \in op, W_{t_{int}}(key) \neq \emptyset \rightarrow \arg\max_{w(t,key) \in W_{t_{int}}(key)} t \prec r(t_{int}, key) \tag{10}$$

■

この定義は，read 命令で読む時刻を明示的に指定するとき，read/write に関する競合による順序関係がヒストリ生成前に定まることを示す．ただし，順序関係が必要な write 命令は式 (9) に基づいて定まり，読まれない過去の write とは直接順序関係が定まらない点に注意する．

以上の定義に基づき，命令集合の *linearizability* を定める．

［定義 4］ 命令の集合 op 及び op 上の順序関係 $\prec$ が与えられたとする．$\prec$ が read-from に基づく順序関係を全て含みかつ半順序であるとき，op は *linearizable* である． ■

つまり，命令集合 op を事前に与えられた read-from の順序関係を壊すことなく何らかの実行ヒストリー（全順序な命令列）で処理できるとき，その命令集合は linearizable である．

最後に，ストリーム処理におけるトランザクションを定める．

［定義 5］ トランザクション $T = (op, \prec)$ は有限な命令集合 op と op 上の linearizable な順序関係 $\prec$ の組である． ■

3.2.3 トランザクション間の順序関係

トランザクション間の順序関係を，各トランザクション間の read-from 関係を用いて定める．

［定義 6］ トランザクションの集合 $\mathcal{T}$ が与えられたとする．トランザクション集合中の全ての命令の集合 $OP = \bigcup_{T \in \mathcal{T}} T.op$ に対し，read-from に基づく OP 上の順序関係を $\prec$ で表す．このとき，各トランザクション間の read-from に基づく順序関係を以下の式に基づき与える．

$$\forall w(t, key) \in T.op, r(t_{int}, key) \in T'.op, w(t, key) \prec r(t_{int}, key) \rightarrow T \prec T' \tag{11}$$

■

つまり，あるトランザクションで書いた値を別のトランザクションで読むことがわかっているとき，それらの間には事前に（実行ヒストリーに配置する前に）順序関係が定まるとする

更に，定義 6 から，以下の定理が導ける．

［定理 1］ Read-from に基づくトランザクション集合 $\mathcal{T}$ 上の順序関係 $\prec$ が半順序であるとき，全トランザクションの命令集合 OP は linearizable である．更に，OP から生成できる実行ヒストリーには，少なくとも一つ serializable なものが存在する． ■

つまり，ある時点におけるトランザクションの集合 $\mathcal{T}$ が与えられたとき，各トランザクション間の順序関係が半順序で表せる

か確認することで，何らかの実行ヒストリーが作成可能であるか確認できる．加えて，トランザクション間の順序関係が半順序であるため，少なくとも一つのシリアルヒストリーが存在する．以降，$\mathcal{T}$ 上の read-from に基づく順序関係が半順序であることを，$\mathcal{T}$ **が** *linearizable* **である**と呼ぶ．

$\mathcal{T}$ が linearizable であるかは，多項式時間で判定できる．全命令集合 OP 上の read-from に基づく順序関係は，任意の read/write 命令の組を確認することで作成できる．その後，命令集合 OP 上の順序関係からトランザクション間の順序関係を導き，それが半順序であるか確認する．半順序であるかはトポロジカルソートなどで循環の有無を確認すればよいため，トランザクションの数とその間の順序関係の数に対して線形で判定できる．つまり，全体としての計算時間は多項式時間に収まる．

4 おわりに

本項では，データストリーム処理におけるトランザクション処理に着目し，ストリーム処理においてトランザクションが満たすべき順序関係について議論した．今後は，検討した内容について議論を進めアイデアを具体化すると共に，理論的な保証やシステムとしての実装を行う予定である．

謝　辞

本研究は，JSPS 科研費（16H01722, 18H06461）の助成，および，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務による．

文　献

[1] Apache Flink 1.7 Documentation: Event Time. `https://ci.apache.org/projects/flink/flink-docs-release-1.7/dev/event_time.html`. Accessed: Feburary 24, 2019.

[2] Apache Flink: Stateful computations over data streams. `https://flink.apache.org/`. Accessed: January 6, 2019.

[3] Apache Hadoop. `https://hadoop.apache.org/`. Accessed: January 6, 2019.

[4] Apache Spark — unified analytics engine for big data. `https://spark.apache.org/`. Accessed: January 6, 2019.

[5] Samza. `http://samza.apache.org/`. Accessed: January 6, 2019.

[6] Spark Streaming | Apache Spark. `https://spark.apache.org/streaming/`. Accessed: January 6, 2019.

[7] I. Botan, P. M. Fischer, D. Kossmann, and N. Tatbul. Transactional stream processing. In *Proc. EDBT*, pages 204–215, 2012.

[8] H. Chen and M. Migliavacca. StreamDB: A unified data management system for service-based cloud application. In *Proc. IEEE SCC*, pages 169–176, 2018.

[9] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. OSDI*, pages 137–150, 2004.

[10] R. Kallman, Y. Zhang, J. Hugg, D. J. Abadi, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, and M. Stonebraker. H-store: A High-Performance, Distributed Main Memory Transaction Processing System. *PVLDB*, 1(2):1496–1499, 2008.

[11] J. Meehan, A. Pavlo, M. Stonebraker, K. Tufte, H. Wang, N. Tatbul, S. Zdonik, C. Aslantas, U. Cetintemel, J. Du, T. Kraska, S. Madden, and D. Maier. S-Store: Streaming meets transaction processing. *PVLDB*, 8(13):2134–2145, 2015.

[12] B. Mozafari, J. Ramnarayan, S. Menon, Y. Mahajan, S. Chakraborty, H. Bhanawat, and K. Bachhav. SnappyData: A unified cluster for streaming, transactions, and interactive analytics. In *Proc. CIDR*, 2017.

[13] G. Weikum and G. Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann Publishers Inc., 1st edition, 2001.

[14] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proc. SOSP*, pages 423–438, 2013.