MorphStream: Scalable Processing of Transactions over Streams

Siqi Xiang* Zhonghao Yang* Jianjun Zhao† Yancan Mao‡ Shuhao Zhang§
*Singapore University of Technology and Design †Huazhong University of Science and Technology
‡National University of Singapore §Nanyang Technological Univeristy
{siqi_xiang, zhonghao_yang}@mymail.sutd.edu.sg, curry_zhao@hust.edu.cn
maoyancan@u.nus.edu, shuhao.zhang@ntu.edu.sg

2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00434

***Abstract*—In the realm of transactional stream processing (TSP), the challenge lies in providing a unified execution model that seamlessly integrates transactional and stream-oriented capabilities. Existing TSP engines (TSPEs) largely employ non-adaptive scheduling techniques, leaving multicore parallelism underutilized due to intricate workload dependencies. We demonstrate `MorphStream`, a state-of-the-art TSPE built for unprecedented scalability on multicores. `MorphStream` distinguishes itself by employing an adaptive scheduling algorithm, explicitly designed to unlock the full potential of multicore architectures even under complex workload conditions. This enables `MorphStream` to make optimal trade-offs in performance metrics under varying workload characteristics. To enhance user engagement, the demonstration will showcase `MorphStream`'s graphical user interface, specifically engineered to simplify the implementation and deployment of complex streaming applications while providing detailed and comprehensive performance monitoring and analytics for the job execution runtime.**

I. Introduction

Transactional stream processing (TSP) can be broadly defined as processing streaming data with correctness guarantees, including those intrinsic to stream processing (such as time order and exactly-once semantics), as well as the `ACID` guarantees found in traditional databases [1]. TSP has gained popularity in recent years due to its ability to support novel applications and system optimizations, with use cases often involving streaming facilities to persist or offer near-real-time views of shared state, while transactional facilities ensure a consistent representation of the shared state. TSP engines (TSPEs) [2], [3], [4], [5] have been proposed, but existing TSPEs mostly rely on locks and simple non-adaptive scheduling strategies, resulting in significant overhead and poor utilization of modern parallel hardware [4]. The scalability limitation of existing TSPEs limits their practicality.

Scaling TSPEs is challenging because of the non-trivial combination of both transaction and stream-oriented properties in TSPEs that lead to complicated workload dependencies. To tackle this challenge, we have developed `MorphStream` [6], a novel TSPE with excellent scalability on multicores. Similar to other TSPEs, `MorphStream` adopts transactional semantics during the processing of continuous data streams, where accesses to the shared state are modelled as state transactions. Different from others, `MorphStream` identifies the fine-grained *temporal*, *logical*, and *parametric* dependencies among state access operations of a batch of state transactions. Then, it maps the workloads with dependencies into a *task precedence graph* (TPG), where vertexes map to state access operations, and edges map to fine-grained dependencies among operations. Based on the TPG, `MorphStream` exploits the full potential of hardware parallelism by decomposing the scheduling strategy into three dimensions of scheduling decisions: 1) structured or non-structured exploration strategies, 2) single operation or group of operations as the unit of scheduling, and 3) lazy or eager abort handling mechanisms. Subsequently, `MorphStream` can adaptively switch to a different scheduling strategy by making suitable scheduling decisions in each dimension, guided by a heuristic-based decision model that analyzes the trade-offs under varying workload characteristics.

Our demo will showcase how `MorphStream` can effectively support novel stream applications involving the processing of transactions over streams. In particular, we cover three novel aspects of `MorphStream`: 1) *Programming APIs:* With `MorphStream`, users can easily express a wide range of novel stream applications that require correctness guarantees of stream processing (such as time order and exactly-once semantics), as well as `ACID` guarantees; 2) *TPG-based Adaptive Scheduling:* `MorphStream` is able to flexibly morph among scheduling strategies, adapting to dynamically changing workload characteristics. Guided by a lightweight decision model, `MorphStream` can make the correct scheduling decision at runtime with minor overheads; 3) *High Performance:* Compared to existing approaches, much of the additional system overhead of `MorphStream` comes from constructing and exploring the TPG concurrently and correctly, considering that input events may arrive *out-of-order*. To reduce TPG construction and exploration overhead, `MorphStream` has a novel highly parallelized system architecture, which ends up with a multi-times performance improvement over the state-of-the-art.

II. Preliminaries

In analogy to conventional transaction processing, TSP guarantees `ACID` properties [7]. In addition, TSP needs to further ensure stream processing properties such that dependent state accesses strictly follow their timestamp sequence [7], [8], [2], [4]. We define a *state access operation*

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00434

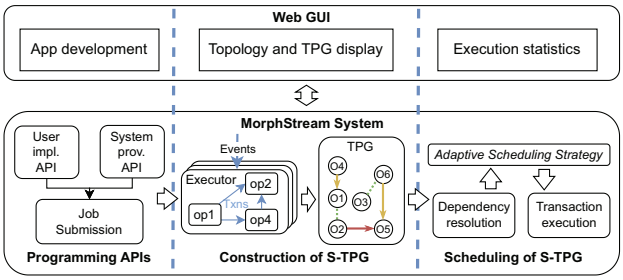


Fig. 1: The key components of MorphStream and its GUI.

as a read or write operation to shared state, denoted as $O_i = Read_{ts}(k)$ or $Write_{ts}(k, v)$. Timestamp ts is defined as the time of its triggering input event, while k denotes the state to read or write, and v denotes the value to write. The key k can be extracted from the input event [4], [2], while v may depend on the value of a list of states, i.e., $v = f(k_1, k_2, ..., k_m)$, where f is a read-only user-defined function. The set of state access operations triggered by the processing of one input tuple is defined as one *state transaction*, denoted as $txn_{ts} = < O_1, O_2, ... O_n >$. Subsequently, a schedule (S) of state transactions txn_{t1}, txn_{t2}, ..., txn_{tn} is correct if it is *conflict equivalent* to $txn_{t1} \prec txn_{t2} \prec ... \prec txn_{tn}$, where $\prec$ means that the left operand precedes the right operand.

To sustain high input stream ingress rates, scaling TSP is essential. Hence, a design goal of TSPEs is to maximize system concurrency while maintaining a correct schedule. However, it is a non-trivial challenge due to complex inter- and intra-dependencies among state transactions. In the following, we summarize three key dependencies among state transactions.

- **Temporal Dependency (TD):** O_i temporally depends on O_j if they are not from the same state transaction, but they access the same state, and O_i has a larger timestamp. Tracking TD enforces that state accesses follow the event sequence.
- **Parametric Dependency (PD):** $O_i = Write(k_i, v)$, where $v = f(k_1, k_2, ..., k_m)$, parametrically depends on $O_j = Write(k_j, v')$ if $k_j \neq k_i$, $k_j \in k_1, k_2, ..., k_m$, and O_i has a larger timestamp. Tracking PD resolves the potential conflicts among write operations due to user-defined functions.
- **Logical Dependency (LD):** O_i and O_j logically depend on each other if they belong to the same state transaction. Tracking LD ensures that the aborting of one operation leads to aborting all operations of the same state transaction.

III. System Overview

MorphStream is a novel TSPE, purposefully engineered for impeccable performance on contemporary parallel hardware, even amidst dynamically shifting and heavily contested workloads. Figure 1 presents a schematic of MorphStream's framework and its GUI. Initiating with the 'App Development' phase, users craft stream applications using user-centric and system-provided APIs, modelled as Directed Acyclic Graphs (DAGs). The system's flexibility is highlighted by its blend of user-defined and system-endorsed APIs, with atomic state access operations encapsulated by system APIs. Central to MorphStream is a stateful Task Precedence Graph (S-TPG). This graph, with vertices representing state access tasks and edges denoting dependencies, is foundational to MorphStream's scheduling. In its TPG-based scheduling, MorphStream emphasizes transactional integrity in TSPEs. Strategies for operation exploration, determining scheduling units, and managing transactional aborts are evident in its execution phase. Compared to peers, MorphStream consistently demonstrates adaptability and superior performance across varied workloads.

A. Programming APIs

In MorphStream, transactional stream applications are depicted as a Directed Acyclic Graph (DAG), adhering to a dataflow model. Within each vertex, MorphStream extends both user-defined and system-integrated APIs to enhance the transaction processing over streams. While users mould the user-defined APIs to fit their specific application requirements, the system-integrated APIs function akin to library procedures. The user-implemented APIs require users to follow a three-step procedure to implement the operations of an operator: Firstly, *preprocessing* the input events to identify the read/write sets of state transactions. Secondly, performing *state access*, where all state accesses are expressed through system-provided APIs using state transactions. Finally, *post-processing* is conducted to further process the input events based on the access results and generate corresponding outputs. The catalog of system-integrated APIs encompasses operations such as $READ$, $WRITE$, and $READ\_MODIFY$, symbolizing the atomic actions of a state transaction. Within MorphStream, a window operation [9] is characterized as either a read or write task, further contextualized by a temporal range and a designated trigger instant. Once an application is fully sculpted, it is dispatched as a job to MorphStream, priming it for the ensuing execution.

B. Construction of S-TPG

Central to MorphStream's design is its advanced capability to facilitate intricate three-dimensional scheduling decisions promptly. This efficacy is achieved by distinctly partitioning dependency resolution and execution into 'stream processing' and 'transaction processing' phases. Such strategic bifurcation not only enables MorphStream to efficiently batch continuous state transactions but also grants the flexibility to recalibrate its scheduling techniques in response to the dynamic nature of the workload. For every batch of state transactions, MorphStream constructs a stateful task precedence graph (S-TPG), wherein vertices symbolize atomic state access operations, and edges establish the dependencies between them. Each vertex in the S-TPG is annotated with

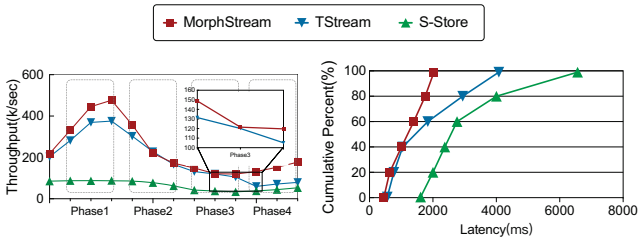
a finite state machine that tracks the execution status of the corresponding operation. The overall scheduling process thus integrates the concurrent formation and three-dimensional strategy considerations applied to the S-TPG.

Constructing the S-TPG with minimal overhead is paramount. Yet, discerning the three dependency types (TDs, LDs, PDs) among state access operations poses challenges, especially when transactions arrive *out-of-order*. To mitigate this, the S-TPG construction is divided into two phases: *a) Stream Processing Phase:* This phase is focused on the identification of dependencies within the same transaction. LDs are recognized based on statement orders. For TDs, operations are organized into key-partitioned lists, chronologically ordered by timestamp and centered around each operation's target state. Additionally, "proxy operations" are maintained for write operations to discern PDs; these are essentially read operations aligned with the keys of the associated write task. *b) Transaction Processing Phase:* Here, subsequent state transactions are temporarily halted until the transition back to the stream processing phase. Efficiently pinpointing TDs and PDs becomes feasible at this juncture by iterating through the chronologically ordered lists and the "proxy operations". This swift and accurate dependency identification, combined with rapid S-TPG construction, empowers MorphStream to adapt its scheduling tactics, aligning with the evolving workload attributes of diverse state transaction batches.

C. Adaptive Scheduling based on S-TPG

In MorphStream, transaction scheduling efficacy depends on managing three types of dependencies via the S-TPG. For each batch of state transactions, the process starts with initializing the S-TPG, establishing the framework of operations and dependencies. Execution then unfolds through two exploratory paths: a structured approach employing depth-first or breadth-first search-like traversals for systematic exploration, or an unstructured approach via random traversal, offering adaptability. Concurrently, threads decide on the scheduling granularity, choosing between single operation or group scheduling to strike a balance between dependency resolution overhead and scheduling scalability. In cases of transaction abort, MorphStream adopts varied strategies, ranging from eager aborts for swift termination of failing operations to lazy aborts for deferred handling. These strategies are dynamically selected and fine-tuned in response to operational outcomes and overall system performance.

To evaluate MorphStream, we compare its performance against two state-of-the-art TSPEs: S-Store [2] and TStream [4]. We use Streaming Ledger [5] (SL) as the base application and divide the workloads into four phases. The first three phases evaluate MorphStream's adaptability to varying workload characteristics, such as deposit transaction density, key skewness, and transfer transaction ratios. For example, in Phase 1, MorphStream leverages an adaptive scheduling algorithm to yield up to 1.27x higher throughput than the closest competitor. As workloads change, MorphStream dynamically adjusts its scheduling strategy, such as shifting from a structured approach to an unstructured approach, to effectively resolve dependencies and maintain high performance. In the fourth experimental phase, we focus on MorphStream's resilience to an increasing ratio of aborting transactions. While the performance of TStream deteriorates due to redo overhead, MorphStream adopts an abort mechanism that evolves in response to changing abort rates, resulting in a throughput 2.2x to 3.4x higher than other systems. Additionally, unlike S-Store and TStream, whose performance declines under changing workloads, MorphStream's adaptive scheduling minimizes tail latency by dynamically optimizing for varied workload characteristics.



(a) Throughput (b) Latency

Fig. 2: Evaluation on Dynamic Workload.

IV. Demonstration Scenarios

We will demonstrate the key features of MorphStream by using SL as an example. The demonstration will be segmented into three critical parts, each focusing on distinct functionalities provided by the MorphStream's GUI.

A. Application Development and Deployment

MorphStream's GUI offers an efficient environment for stream application development. A frontend code editor is designed for ease in implementing transactional stream processing applications as shown in Figure 3. With built-in syntax highlighting, auto-suggestions, and integrated MorphStream's client-side API documentation, the user can easily define customized streaming applications with transactional guarantees. This hands-on interface ensures participants gain both theoretical insight and practical proficiency in using MorphStream for high-performance stream processing. The submitted code will be subsequently compiled as a new streaming job deployed in MorphStream.

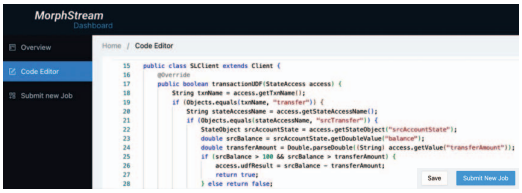


Fig. 3: App development & deployment in MorphStream.

B. Display of Topology and TPG

Figure 4 showcases the GUI's display of application topology and S-TPG. The application topology consists of all operators integral to the stream-oriented functionalities of MorphStream. The S-TPG, as the key component that participates in MorphStream's adaptive scheduling process, visualizes the fine-grained transactional dependencies among state access operations. Each node represents one state access, and three types of dependencies (TD, LD, and PD) are visualized as color-coded edges for clarity. Detailed state access information for each operation is revealed upon hovering over it with the cursor. Adjacent to the S-TPG, the lower right section provides aggregated numbers and ratios of TD, LD, and PD on the current S-TPG. These statistics offer users valuable insights into the heuristic-based decision model that governs MorphStream's adaptive scheduling, aiding in performance optimization or troubleshooting.

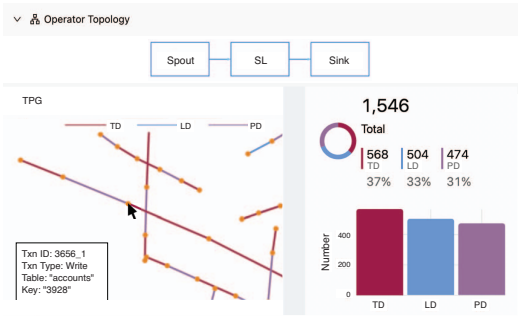


Fig. 4: Display of Application Topology and TPG statistics.

C. Display of Runtime Execution Statistics

We highlight the GUI's ability to display runtime execution statistics in Figure 5. The interface dynamically presents MorphStream's throughput and latency for each batch, enabling users to make instant performance-enhancing adjustments. Furthermore, the GUI offers a granular breakdown of execution durations for each input batch, segmented into: 1) Useful Time, reserved for the actual time spent in transaction execution, 2) Construction Time, designated for S-TPG creation, 3) Exploration Time, allocated for exploring the optimal transactional scheduling strategy, and 4) Abort Time, consumed during transaction abort handling. These fine-grained metrics furnish a clear view of MorphStream's execution efficiency, proving invaluable for troubleshooting and performance optimization. Additionally, the interface summarizes comprehensive statistics for every batch of input events, detailing metrics such as the statistical summary of event processing latency, overall throughput, and the chosen scheduling strategy. Runtime statistics of different input batches are dynamically updated in the GUI, and users can also review the historical performance data of any previous batches.

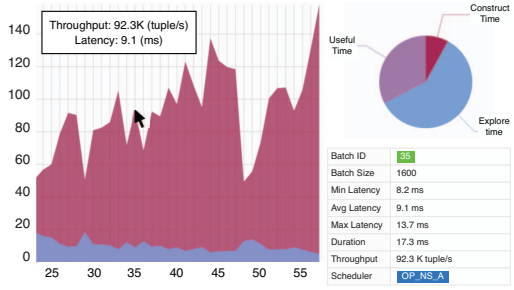


Fig. 5: Display of Runtime Statistics.

V. Summary

None of the existing TSPEs can maximize performance under different and dynamically changing workload characteristics, which limits their practicality. In this demonstration, we aim to show that MorphStream can achieve scalable processing of transactions over streams. We plan to showcase MorphStream's ability to dynamically adjust its scheduling strategy by exploring the three scheduling dimensions based on the analysis of decision trade-offs under different workload characteristics. Additionally, this approach results in better overall performance than using a single scheduling strategy. The audience will have the opportunity to interact with MorphStream and gain a better understanding of *how to use it*, *its unique features*, and *its performance*.

Acknowledgement

This work is supported by a MoE AcRF Tier 2 grant (MOE-T2EP20122-0010), the National Research Foundation, Singapore and Infocomm Media Development Authority under its Future Communications Research & Development Programme FCP-SUTD-RG-2022-005 & FCP-SUTD-RG-2022-006 and a startup grant of NTU (023452-00001). Corresponding author is Shuhao Zhang.

References

[1] S. Zhang, J. Soto, and V. Markl, "A survey on transactional stream processing," *The VLDB Journal*, Sep 2023.

[2] J. Meehan, N. Tatbul, and et al., "S-store: Streaming meets transaction processing," *Proc. VLDB Endow.*, sep 2015.

[3] L. Affetti, A. Margara, and G. Cugola, "Tspoon: Transactions on a stream processor," *JPDC*, vol. 140, pp. 65–79, 2020.

[4] S. Zhang, Y. Wu, and et al., "Towards concurrent stateful stream processing on multicore processors," in *ICDE*, 2020.

[5] S. A. Transactions and S. Data, "Data Artisans Streaming Ledger Serializable ACID Transactions on Streaming Data, https://www.da-platform.com/streaming-ledger," 2018.

[6] Y. Mao, J. Zhao, and et al., "Morphstream: Adaptive scheduling for scalable transactional stream processing on multicores," in *SIGMOD*, 2023.

[7] L. Affetti, A. Margara, and G. Cugola, "Flowdb: Integrating stream processing and consistent state management," in *DEBS*, 2017.

[8] U. Cetintemel, J. Du, and et al., "S-store: A streaming newsql system for big velocity applications," *Proc. VLDB Endow. 2014*.

[9] L. Golab, K. G. Bijay, and M. T. Özsu, "On concurrency control in sliding window queries over data streams," in *EDBT*, 2006.