Fd-Stream: Feedback-Driven Approximate Fault Tolerance for Stateful Stream Computing

Yinuo Fan[1], Dawei Sun[1*], Zhaojun Wang[1], Jia Peng[1], Shang Gao[2], Rajkumar Buyya[3]
[1]School of Artificial Intelligence, China University of Geosciences, Beijing, 100083, China
fanyinuocn@email.cugb.edu.cn; sundaweicn@cugb.edu.cn; wangzhaojun@email.cugb.edu.cn; pengjia@email.cugb.edu.cn
[2]School of Information Technology, Deakin University, Waurn Ponds, Victoria, 3216, Australia
shang.gao@deakin.edu.au
[3]Quantum Cloud Computing and Distributed Systems (qCLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Grattan Street, Parkville, Victoria, 3010, Australia
rbuyya@unimelb.edu.au

***Abstract*—Fault tolerance plays a critical role in achieving high reliability in stateful stream computing systems. Approximate fault tolerance reduces state backup and failure recovery costs by allowing controlled output errors. However, existing approximate fault tolerance methods lack quantitative modeling of error propagation across operators and runtime feedback regulation, resulting in limited adaptability to error fluctuations. To overcome these limitations, we propose Fd-Stream, a feedback-driven approximate fault tolerance framework with three key components: (1) an anchor-guided global error estimation method that obtains global output errors through anchor sampling and continuously tracks runtime accuracy variations without requiring full exact comparisons; (2) a differentiated sampling-rate control mechanism that dynamically adjusts the backup sampling rates of stateful operators based on global error feedback and operator importance, reducing redundant backup overhead when the error remains within a safe range; (3) a feedback-assisted approximate state recovery strategy that rapidly reconstructs operator states after failures, avoiding costly full-state recovery and data replay. We implement Fd-Stream on Apache Storm and evaluate its effectiveness through extensive experiments. Results show that Fd-Stream reduces failure recovery latency by up to 59.03% and memory utilization by up to 29.71% compared with state-of-the-art methods.**

***Index Terms*—feedback-driven, approximate fault tolerance, stateful operator, stream computing**

I. Introduction

Stream computing systems have become essential infrastructure for latency-sensitive applications such as online advertising and financial risk control [1], [2]. To support stateful operations such as window aggregation and join queries, these systems continuously maintain and update large amounts of operator state during execution [3], [4]. However, failures caused by process crashes, network anomalies, and resource contention, failures may occur during system execution, causing the loss or inconsistency of in-memory operator states, which in turn leads to result errors and service interruptions [5]. To address this challenge, many studies have investigated fault tolerance techniques for stream processing, aiming to recover or reconstruct operator states after failures and to ensure output reliability, state recoverability, and continuous service availability [6].

Existing fault tolerance methods for stream computing systems mainly rely on strongly consistent recovery techniques, including state checkpointing, active backup, and data replay [7], [8]. These methods restore the system to a consistent state after failures by periodically persisting operator states, recording input data lineage, or maintaining redundant execution replicas, while providing processing semantics such as exactly-once or at-least-once guarantees [9]. However, they often incur substantial runtime and recovery overheads. Specifically, state checkpointing increases the costs of serialization, storage, and network transmission; active backup continuously consumes additional computing resources; and data replay during recovery may cause queue congestion and prolonged recovery latency [10], [11]. In high-throughput and state-heavy scenarios, such overheads can significantly reduce the real-time processing capability of the system.

Approximate fault tolerance provides a lightweight alternative to strongly consistent fault tolerance [12], [13]. By allowing bounded output errors, it reduces the costs of state persistence and failure recovery through sampled backup and approximate state recovery [14]. As a result, it is particularly suitable for applications that place greater emphasis on result timeliness [15]. However, existing approximate fault tolerance methods still face two key challenges: (1) they lack a low-cost mechanism for observing global output errors, making it difficult to promptly capture runtime accuracy changes and incorporate them back into fault tolerance decisions; and (2) they lack a dynamic sampling-rate control mechanism that accounts for the heterogeneity of stateful operators, making it difficult to allocate backup resources according to operator importance and error propagation impact. As a result, these methods cannot effectively suppress error fluctuations under failures or workload changes. They may also lead to insufficient protection for operators that have a large impact on the final output (critical operators), while introducing redundant backups for operators with relatively small impact on the final output (marginal operators).

To address these limitations, we proposes Fd-Stream, a feedback-driven approximate fault tolerance framework for stateful stream computing. Fd-Stream senses global output

errors through anchor sampling, dynamically adjusts backup sampling rates based on operator importance, and rapidly reconstructs states from approximate backups after failures. In this way, it reduces fault tolerance overhead and recovery latency while keeping output errors under control. The main contributions of this paper are as follows:

- An anchor-guided global error estimation method is proposed to estimate the global mean relative error at low cost by maintaining the exact and approximate values of a small number of anchor keys, thereby providing runtime observation signals for feedback regulation.
- A differentiated sampling-rate control mechanism is proposed to dynamically adjust the backup sampling rates of stateful operators by combining global error feedback with operator importance, reducing redundant backup overhead for marginal operators while preserving the accuracy of critical operators.
- A feedback-assisted approximate state recovery strategy is proposed to rapidly restore the states of failed operators from approximate backups and compensate for recovery deviations through subsequent feedback regulation, thereby avoiding costly full-state recovery and data replay.

The rest of this paper is organized as follows: Section II reviews related work. Section III introduces the system models and problem formulation. Section IV describes Fd-Stream's architecture and algorithms. Section V evaluates the performance. Finally, Section VI concludes the paper and discusses future directions.

II. Related Work

A. Fault Tolerance Methods

Many studies have focused on strongly consistent fault tolerance, among which checkpointing is one of the most widely used approaches. For example, Optimizing Checkpoint-Based Fault Tolerance [14] studies the impact of checkpoint intervals on system utilization and latency. DACM [10] dynamically adjusts checkpoint intervals through reinforcement learning, while Checkmate [16] compares the performance of coordinated, uncoordinated, and communication-induced checkpointing protocols under different workloads. A-FP4S [11] partitions state into fragments and restores them in parallel to reduce large-scale state recovery latency. Trisk [17] and DRRS [6] focus on reducing the impact of state migration on online processing. These methods improve the availability of stream computing systems during failure recovery. However, most of them still rely on exact state recovery and do not relax exact recovery semantics to reduce state backup costs.

B. Approximate Fault Tolerance

Approximate fault tolerance reduces state backup and failure recovery costs by allowing controlled output errors. AF-Stream [12] performs backups only when the deviation of states or data items exceeds a predefined threshold, thereby achieving a trade-off between performance and post-failure accuracy while providing bounded error guarantees. WAFP [18] dynamically generates approximate backup ratios according to task input/output rates and backup overhead, so as to reduce fault tolerance latency under a given output accuracy constraint. Local Recovery and Partial Snapshot [19] decomposes a global snapshot into regional snapshots and restores only the affected subset of operators after a failure, thereby reducing the dependency and pause overhead caused by global recovery.

TABLE I
Comparison of Related Methods and Fd-Stream.

Method	Priority	Dimensions	
		Global feedback	Operator awareness
DRRS [6]	Timeliness	✓	×
DACM [10]	Accuracy	✓	×
A-FP4S [11]	Timeliness	×	✓
AF-Stream [12]	Timeliness	✓	×
Checkmate [16]	Accuracy	×	×
Fd-Stream (Ours)	Timeliness	✓	✓

Despite their effectiveness, existing approximate fault tolerance methods still have two limitations. First, they generally lack a lightweight mechanism for estimating global output error at runtime, making it difficult to directly use output accuracy as a feedback signal for fault tolerance regulation. Second, they lack operator-aware control that jointly considers error propagation impact and resource cost when allocating backup resources. Motivated by these limitations, we propose Fd-Stream. By estimating runtime global error through anchor sampling and dynamically adjusting backup sampling rates based on operator importance, Fd-Stream achieves a balance between output accuracy and fault tolerance overhead. Table I summarizes the differences between Fd-Stream and related approaches.

III. System Model and Problem Formulation

A. System Model

Operator state model. A stream application can be represented as a directed acyclic graph (DAG) $G = \{V(G), E(G)\}$, where $V(G) = \{v_i | i \in [1, n]\}$ denotes the set of operators that represent specific processing logic, including both stateful operators $V_s(G)$ and stateless operators $V_u(G)$. $E(G) = \{e_{i,j} | i, j \in [1, n], i \neq j\}$ denotes the set of edges that represent data transmission between operators. Operator state refers to the intermediate computation results maintained by a stateful operator to support continuous stream processing.

For any stateful operator $v_i \in V_s$, assume that its internal state at time t is denoted as $ST_{v_i}(t)$, and its state update function is denoted as $f_{\text{up}}(\cdot)$. At time $t \in W$, a user-defined time window, operator v_i continuously updates its state as input tuples arrive. For arriving tuples τ_k at time $t_k \in W$, the state update of operator v_i is shown in Eq. (1):

$$ST_{v_i}(t_k) = f_{\text{up}}\big(ST_{v_i}(t_{k-1}), \tau_k\big), \tag{1}$$

where $t_{k-1} \in W$ denotes the previous time point of t_k.

State recovery model. Stateful operators are typically deployed as parallel instances across different computing nodes. If a computing node hosting operator v_i fails at time t_f, the

true in-memory state $ST^*_{v_i}(t_f)$ maintained by that operator v_i may be lost. If a strongly consistent recovery method is used, it usually involves several stages, including failure detection, task restart, state loading, and data replay, and the total recovery time T_{total} of operator v_i can be calculated by Eq. (2):

$$T^{\text{total}}_{v_i} = \Delta t^{fd}_{v_i} + \Delta t^{ts}_{v_i} + \Delta t^{sl}_{v_i} + \Delta t^{dr}_{v_i}, \tag{2}$$

where $\Delta t^{fd}_{v_i}$, $\Delta t^{ts}_{v_i}$, $\Delta t^{sl}_{v_i}$ and $\Delta t^{dr}_{v_i}$ denote the latency of failure detection, task restart, state loading, and data replay, respectively. In contrast, approximate fault tolerance directly restores state from approximate backups, avoiding the time overhead caused by complete state loading and historical data replay.

Approximate state refers to an approximate representation of the complete operator state, obtained by selectively backing up part of the state information instead of preserving the full exact state. Approximate state recovery reads the approximate state $\hat{ST}_{v_i}(t_f)$ from external storage and resumes execution based on it, thereby reducing state recovery latency. However, approximate recovery introduces a deviation $\epsilon_{v_i}(t_f)$ between the recovered state and the true state before the failure, as described in Eq. (3):

$$\epsilon_{v_i}(t_f) = f_{\text{diff}}(ST^*_{v_i}(t_f), \hat{ST}_{v_i}(t_f)), \tag{3}$$

where $f_{\text{diff}}(\cdot)$ denotes the state difference metric function. This local recovery error propagates downstream operators along the data transmission direction and affects the final output results. We use output impact degree $O(v_i)$ to quantify the potential influence of the local state error of operator v_i on the final outputs after error propagation through downstream paths.

B. Problem Formulation

Operator importance. State size, topological position, and runtime workload vary significantly across operators. FdStream measures the importance of all stateful operators based on output impact degree, structural dependency, and runtime overhead, so as to identify critical operators and marginal operators. Let $V_{\text{ter}} \in V(G)$ denote the set of terminal operators, $ds(v_i)$ denote the set of downstream operators of v_i, and $\delta \in (0, 1)$ denote the spatial decay factor. Then, the output impact degree $O(v_i)$ can be calculated by Eq. (4):

$$O(v_i) = \begin{cases} 1, & v_i \in V_{\text{ter}}, \\ \sum_{v_j \in ds(v_i)} \delta \cdot O(v_j), & \text{otherwise}. \end{cases} \tag{4}$$

$O(v_i)$ indicates that the closer operator v_i is to the output end and the larger its downstream influence range is, the more likely its local error is to affect the final results.

The structural dependency degree $D(v_i)$ describes the convergence and distribution roles of operator v_i in the stream application, as shown in Eq. (5):

$$D(v_i) = \omega_{in} \cdot D_{in}(v_i) + \omega_{out} \cdot D_{out}(v_i), \tag{5}$$

where $D_{in}(v_i)$ and $D_{out}(v_i)$ denote the in-degree and out-degree of v_i, respectively, i.e., the number of incoming edges from upstream operators and outgoing edges to downstream operators. ω_{in} and ω_{out} are weight parameters. A larger ω_{in} emphasizes operators that aggregate inputs from multiple upstream operators, while a larger ω_{out} emphasizes operators that distribute outputs to multiple downstream operators. The runtime cost $C(v_i)$ describe the current resource pressure of operator v_i through CPU utilization $U_{\text{cpu}}(v_i)$, memory utilization $U_{\text{mem}}(v_i)$, and I/O utilization $U_{\text{io}}(v_i)$, as shown in Eq. (6):

$$C(v_i) = U_{\text{cpu}}(v_i) + U_{\text{mem}}(v_i) + U_{\text{io}}(v_i). \tag{6}$$

Since the three metrics have different dimensions, FdStream applies min-max normalization to each metric. For any metric $X(v_i) \in \{O(v_i), D(v_i), C(v_i)\}$, the normalized value can be calculated by Eq. (7):

$$\bar{X}(v_i) = \frac{X(v_i) - \min_{v_j \in V_s} X(v_j)}{\max_{v_j \in V_s} X(v_j) - \min_{v_j \in V_s} X(v_j) + \epsilon}, \tag{7}$$

where ϵ is a small constant to avoid division by zero. Subsequently, the importance $I(v_i)$ of operator v_i can be calculated by Eq. (8):

$$I(v_i) = \bar{O}(v_i) + \bar{D}(v_i) + \bar{C}(v_i). \tag{8}$$

A larger $I(v_i)$ indicates that the operator v_i has a greater impact on global output accuracy or recovery overhead (critical operator). Therefore, when the global error exceeds a user-defined threshold, critical operators backup sampling rate should be increased with higher priority. Conversely, for marginal operators, the backup sampling rate can be preferentially reduced to release redundant overhead.

Error propagation. Sampled backup means that operator v_i writes only selected state updates to external backup storage according to its backup sampling rate r_{v_i}. In approximate fault tolerance, sampled backup introduces the state deviation of an individual stateful operator (local state errors). For any stateful operator v_i, let N_{v_i} denote the total number of unit state updates generated by input tuples within the time window W, and let r_{v_i} denote the backup sampling rate of v_i. Then, the variance of the approximate state estimator $\text{Var}_{\text{local}}(ST_{v_i})$ recovered from sampled backup can be calculated by Eq. (9):

$$\text{Var}_{\text{local}}(ST_{v_i}) = N_{v_i} \left(\frac{1}{r_{v_i}} - 1 \right). \tag{9}$$

Local errors do not remain confined within a single operator, but continue to propagate along the data flow. Let $us(v_j)$ denote the set of upstream operators of operator v_j, and let $g_{v_j}(\cdot)$ denote the state computation function of v_j. Then, the state of v_j can be abstracted as a function of its upstream states. According to the first-order error propagation approximation [20], the accumulated error variance $\text{Var}_{\text{total}}(ST_{v_i})$ at v_j can be calculated by Eq. (10):

$$\begin{aligned} \text{Var}_{\text{total}}(ST_{v_j}) \approx & \sum_{v_i \in us(v_j)} \left(\frac{\partial g_j}{\partial ST_{v_i}} \right)^2 \text{Var}_{\text{total}}(ST_{v_i}) \\ & + \text{Var}_{\text{local}}(ST_{v_j}), \end{aligned} \tag{10}$$

where the first term represents the accumulated impact of upstream errors after being propagated to the current operator v_j. For each upstream operator v_i, $\mathrm{Var}_{total}(ST_{v_i})$ denotes its accumulated error variance, and $\frac{\partial g_j}{\partial ST_{v_i}}$ denotes the sensitivity of v_j's state computation to the state variation of v_i. Thus, this partial derivative can be regarded as the error amplification coefficient along edge $e_{i,j}$. The second term $\mathrm{Var}_{\mathrm{local}}(ST_{v_j})$ denotes the local state error introduced by the sampled backup of v_j itself.

IV. Fd-Stream: Architecture and Algorithms

A. System Architecture

Fd-Stream is built on Apache Storm, one of the mainstream stream computing systems. Fd-Stream's system architecture is shown in Fig. 1. After the user submits a topology to the system, Nimbus is responsible for task assignment, resource scheduling, and task reassignment after failures, while ZooKeeper is responsible for cluster state coordination, heartbeat monitoring, and failure detection. Feedback-driven approximate fault tolerance consists of three main modules: Global error estimation, Sampling-rate control and Approximate state recovery.

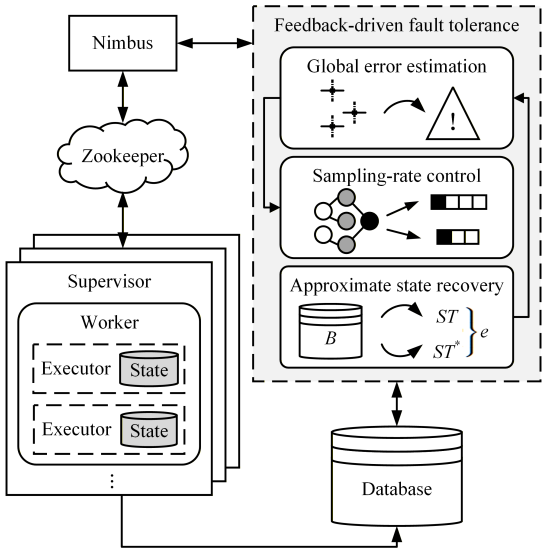

Fig. 1. Fd-Stream's architecture.

Global error estimation selects a small number of anchor keys from the input stream by a deterministic hash rule and maintains two views containing the exact and approximate results of these anchor keys: the exact view and the approximate view. The system compares the relative deviation between the two views, thereby estimating the global output error.

Sampling rate control performs differentiated adjustment based on error feedback and operator importance. It adjusts the sampling rates of critical and marginal operators differently according to different error feedback, thereby balancing output accuracy and backup overhead.

Approximate state recovery module reads the sampled backup state of the failed operators from the external database and uses it as the initial state. Subsequently, the system reduces the impact of the approximately recovered state on the final output error through feedback regulation.

B. Anchor-Guided Global Error Estimation

To avoid the computation and storage overheads introduced by exact comparison over all operator states, Fd-Stream designs an anchor-guided global error estimation method, which estimates the current global output error of the system at low cost. Specifically, Fd-Stream computes a deterministic hash value $h(a)$ for each input key a and normalizes it to $[0, 1)$. If $h(a) < \rho$, the key is selected as an anchor key, where ρ denotes the anchor sampling ratio.

For these anchor keys, the system maintains two views simultaneously: a exact view M^*, which records the true results of the anchor keys, and an approximate view $\hat{M}$, which records the approximate results produced by the sampled-backup of the approximate fault tolerance. A feedback period refers to the time interval between two consecutive rounds of global error estimation and sampling rate adjustment. At the end of each feedback period, the terminal operators compares the results of the same anchor keys in the two views and calculates the global mean relative error $\hat{E}(k)$, as shown in Eq. (11):

$$\hat{E}(k) = \frac{1}{|A_k|} \sum_{a \in A_k} \frac{|\hat{y_a} - y_a^*|}{y_a^*}, \tag{11}$$

where a denote an anchor key, and A_k represent the anchor set in the k-th feedback period. $\hat{y_a}$ and y_a^* are the approximate value and true value of a, respectively. Since the anchor set is much smaller than the full state size of a stream topology, this method estimates the runtime output error of the current topology with low overhead and provides the feedback signal for subsequent differentiated sampling-rate control.

C. Differentiated Sampling-Rate Control

The backup sampling rate of an operator determines the proportion of its state written to external backup storage. A higher sampling rate enables the recovered state after a failure to be closer to the true state, but it also incurs higher serialization, network transmission, and storage overheads. A lower sampling rate reduces system overhead, but it may increase recovery errors. Therefore, Fd-Stream dynamically adjusts the backup sampling rates of different stateful operators according to runtime error variations.

Let the estimated global output error of the stream topology in the k-th feedback period be $\hat{E}(k)$, and let the user-defined maximum error threshold be E_{max}. To avoid frequent sampling-rate updates caused by small error fluctuations, Fd-Stream defines a hold region $[\mu E_{max}, E_{max}]$, where $0 < \mu < 1$. When the error lies within the hold region, the system keeps the current sampling rate unchanged; when the error exceeds E_{max}, the system increases the sampling rate to strengthen state protection; and when the error falls below μE_{max}, the system reduces the sampling rates of marginal operators to release redundant backup resources. The corresponding

sampling-rate adjustment step $\Delta U(k)$ can be calculated by Eq. (12):

$$\Delta U(k) = \begin{cases} K_p\big(\hat{E}(k) - E_{\max}\big) + \beta, & \hat{E}(k) > E_{\max}, \\ 0, & \mu E_{\max} \le \hat{E}(k) \le E_{\max}, \\ -\beta, & \hat{E}(k) < \mu E_{\max}. \end{cases} \tag{12}$$

where K_p denotes the proportional gain coefficient, and β denotes the base adjustment step size. Applying a uniform adjustment to all operators solely according to $\Delta U(k)$ would ignore operator differences. Therefore, Fd-Stream incorporates operator importance $I(v_i)$ for differentiated control. Let $r_{v_i}(k)$ denote the backup sampling rate of operator v_i in the k-th period. The updated sampling rate can be calculated by Eq. (13):

$$r_{v_i}(k+1) = \text{clip}\big(r_{v_i}(k) + \alpha_i(k) \cdot \Delta U(k), r_{\min}, r_{\max}\big), \tag{13}$$

where r_{min} and r_{max} denote the lower and upper bounds of the sampling rate, respectively. The function clip$(\cdot)$ is used to restrict the sampling rate within the range $[r_{min}, r_{max}]$, and $\alpha_i(k)$ denotes the adjustment weight assigned to operator v_i in the current feedback period.

Fd-Stream determines $\alpha_i(k)$ according to the regulation region. When $\hat{E}(k) > E_{max}$, the system selects critical operators in descending order of operator importance and assigns them higher adjustment weights, so that the positive step size is preferentially applied to operators with greater impact on the global error, thereby accelerating error convergence. When $\hat{E}(k) < \mu E_{max}$, the system selects marginal operators in ascending order of operator importance and decreases their sampling rates to reduce the redundant backup overhead of operators with limited impact on output errors. When the error lies within the hold region, the adjustment weights of all operators are set to 0, and the system does not update the sampling rates.

Through the above mechanism, Fd-Stream maps global error feedback to operator-level sampling-rate adjustment. On the one hand, when the estimated global error $\hat{E}(k)$ exceeds the user-defined maximum tolerable error threshold E_{max}, critical operators are protected with higher priority to suppress further error propagation. On the other hand, When $\hat{E}(k)$ falls below the lower bound μE_{max} of the hold region, backup resources of marginal operators are preferentially released by decreasing their sampling rates. In this way, Fd-Stream dynamically balances output accuracy and fault-tolerance overhead.

D. Feedback-Assisted Approximate State Recovery

To avoid the high recovery latency caused by loading complete state and replaying historical data, we proposed a feedback-assisted approximate state recovery. Assume that a stateful operator v_i fails at time t_f, and its true in-memory state $ST^*_{v_i}(t_f)$ is lost. Fd-Stream reads the approximate backup state $\hat{ST}_{v_i}(t_f)$ of this operator from external storage and uses it as the recovered initial state: $ST_{v_i}(t_{rec}) \leftarrow \hat{ST}_{v_i}(t_f)$, where t_{rec} denotes the time when the operator completes recovery. Let $R = \{r_{v_i} \mid v_i \in V_s\}$ denote the backup sampling rate configuration of all stateful operators in the current stream topology, where r_{v_i} represents the backup sampling rate of stateful operator v_i. R' denotes the updated sampling rate configuration obtained after feedback regulation.

To reduce the impact of the deviation between the approximate recovered state and the pre-failure true state on final topology outputs, Fd-Stream performs feedback regulation after the operator resumes execution. Here, feedback regulation refers to using the estimated global error $\hat{E}(k)$ to trigger differentiated sampling-rate control and update the sampling-rate configuration R'. This allows newly arriving data to update the state under the adjusted backup configuration, thereby gradually reducing the impact of recovery deviations.

Algorithm 1: Feedback-assisted approximate state recovery.

Input: Operator v_i, current time t, failure time t_f, backup storage B, initial protection window T_{init}, upper bound of the sampling rate r_{max}, error threshold E_{max}, backup sampling rate set R, and current error estimate $\hat{E}(k)$;
Output: Recovered initial state $ST_{v_i}(t_{rec})$.

```
Set R' ← R;
if t < T_init then
    foreach v_i ∈ V_s do
        Set r_{v_i} ← r_max in R';
    end
end
if v_i fails at t_f then
    Read approximate backup state Ŝ T_{v_i}(t_f) from B;
    Set ST_{v_i}(t_rec) ← Ŝ T_{v_i}(t_f);
end
if Ê > E_max then
    Update R' using the differentiated sampling-rate
      control mechanism;
end
else
    Keep R' unchanged;
end
return ST_{v_i}(t_rec);
```

In addition, during the system cold-start phase, external backup states have not yet been sufficiently accumulated, and error signals are also unstable. To avoid low recovery quality caused by insufficient backups in the initial phase, Fd-Stream sets the sampling rates of stateful operators to the upper bound r_{max} within the initial protection window T_{init}. After the system passes this protection window, it enters the regular feedback regulation process. This strategy avoids full state recovery and historical data replay after a failure, transforming the recovery process from "exact rollback" into "approximate recovery followed by subsequent compensation." As a result,

Fd-Stream can reduce failure recovery latency while maintaining controllable output errors through feedback regulation.

The workflow of the feedback-assisted approximate state recovery is outline in Algorithm 1. The input consists of the failed operator, the current time t, the failure time t_f, the external backup storage B, the initial protection window T_{init}, the upper bound of the sampling rate r_{max}, the error threshold E_{max}, the current global error estimate $\hat{E}$, and the current sampling rate configuration $\mathbf{R}$. The output is the recovered operator state $ST_{v_i}(t_{rec})$. Step 1 initializes $\mathbf{R}'$ as the current sampling rate configuration $\mathbf{R}$. Steps 2-5 determine whether the system is within the initial protection window T_{init} and whether backup sampling rates have been configured for stateful operators. Steps 7-10 read the approximate backup state $\hat{ST}_{v_i}(t_f)$ of the operator from the external backup storage B, and use it as the initial state $ST_{v_i}(t_{rec})$ at the recovery time t_{rec}. Steps 11-16 determine whether post-recovery error compensation is required according to the current global error estimate $\hat{E}$. The time complexity of Algorithm 1 is $O(n)$, where n denotes the number of all stateful operators.

V. Performance Evaluation

We evaluate system performance using three metrics: resource utilization, failure recovery latency, and end-to-end processing latency.

A. Experimental Environment

Cluster setup. The experimental cluster consists of 32 computing nodes, each equipped with a uniform hardware configuration of a 2-core CPU, 4 GB of memory, and a 50 GB disk. One node serves as the master node running Nimbus, one node serves as the coordination node running Zookeeper, and the remaining 30 nodes serve as worker nodes running Supervisor and Redis storage instances. The software configuration used in the experiments is shown in Table II.

Dataset and benchmark. We evaluate Fd-Stream using the representative stream application: Yahoo Streaming Benchmark (YSB) [21] with the public dataset Yahoo Webscope S5 Dataset [22]. The topology structures and parallelism configurations are shown in Fig. 2.

TABLE II
Software Configuration.

Software	Version
OS	Ubuntu 24.04.2 64 bit
Apache Storm	Apache-storm-2.8.0
JDK	Jdk-21.0.2 64 bit
Apache Zookeeper	Apache-Zookeeper-3.5.7
Python	Python 3.8.10
Redis	Redis-7.0.15

Baseline. The experiments select two representative fault tolerance mechanisms for stream computing as comparative baselines: (1) Full Backup [23], a strongly consistent fault tolerance mechanism for stream computing. This mechanism periodically serializes the complete internal states of all stateful operators and writes them to external storage at a fixed interval or frequency. (2) WAFP [18], a load-aware approximate fault tolerance strategy that can allocate approximate backup ratios according to load imbalance.

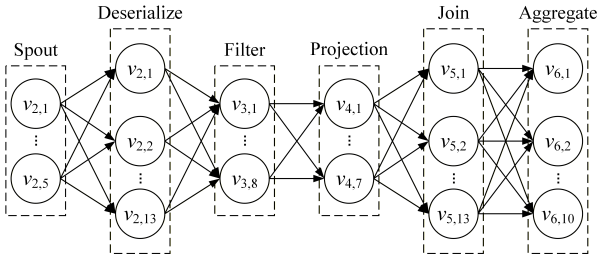

Fig. 2. Topology structure and parallelism configuration of YSB.

B. Resource Utilization

This experiment evaluates the resource utilization of Fd-Stream and the baseline methods, including network I/O overhead, CPU utilization, and memory utilization. The workload is uniformly set to 20,000 tuples/s, and the average resource consumption during the stable running phase of the system is recorded.

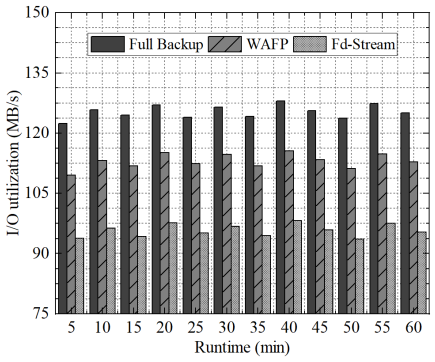

Fig. 3. I/O utilization of Fd-Stream and baselines.

As shown in Fig. 3, Full Backup continuously writes the complete operator states, resulting in an average I/O overhead of 125.3 MB/s. WAFP reduces part of the state write volume through approximate backup, lowering the average I/O overhead to 113.0 MB/s. Fd-Stream uses global error feedback and operator importance to regulate the sampling rates, preferentially reducing redundant backups for marginal operators and further decreasing the average I/O overhead to 95.7 MB/s. Compared with Full Backup, Fd-Stream reduces the I/O overhead by approximately 23.6%; compared with WAFP, it achieves a reduction of approximately 15.3%.

As shown in Fig. 4, the average CPU utilization of Full Backup is 82.9%, with the main overhead coming from full-state serialization and frequent persistence. The average CPU utilization of WAFP is 75.8%. In contrast, Fd-Stream achieves an average CPU utilization of 62.4%, which is lower than

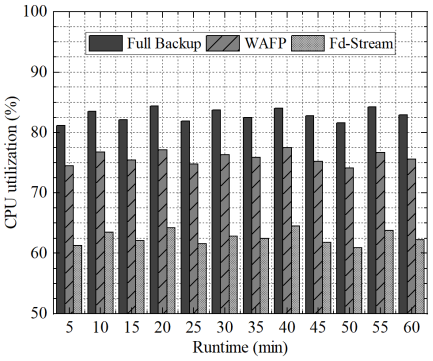

Fig. 4. CPU utilization of Fd-Stream and baselines.

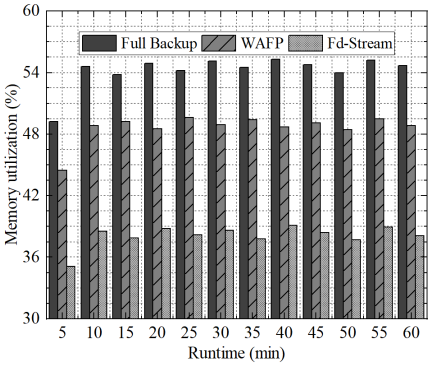

Fig. 5. Memory utilization of Fd-Stream and baselines.

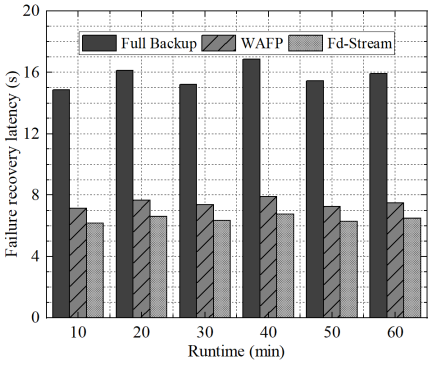

Fig. 6. Failure recovery latency of Fd-Stream and baselines.

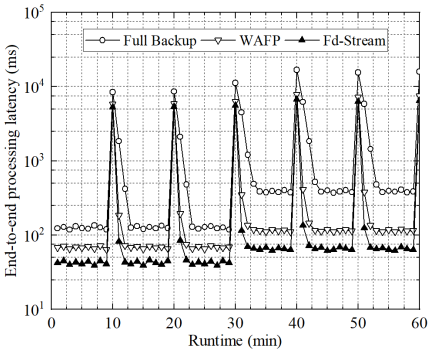

Fig. 7. End-to-end processing latency of Fd-Stream and baselines.

both baseline methods. This is because that Fd-Stream reduces unnecessary state backups and serialization operations, thereby lowering computational resource consumption.

As shown in Fig. 5, Full Backup needs to maintain complete state replicas, resulting in an average memory utilization of 73.8%. WAFP reduces the memory utilization to 66.7%. Fd-Stream further reduces the average memory utilization to 54.2%, achieving a reduction of approximately 18.7% compared with WAFP. This is because Fd-Stream preferentially lowers the sampling rates of low-importance operators when the error remains within a safe range, thereby reducing the continuous accumulation of non-critical states.

C. Failure Recovery Latency

Failure recovery latency measures the time required for the system to restore its processing capability after a Worker process failure. In the experiment, the workload is set to 40,000 tuples/s, and a Worker process hosting stateful operators is forcibly terminated every 10 minutes to simulate node failures. The recovery latency is measured from the time when the process is terminated to the moment when the system completes failure detection, task rescheduling, operator state recovery, and successfully processes the first newly arrived tuple.

As shown in Fig. 6, Full Backup has an average failure recovery latency of 15.728 s. Since this strategy needs to load the complete state and process the data accumulated during the failure period, it tends to incur high deserialization overhead and queue congestion during recovery. WAFP reduces the size of the recovered state through approximate backup, lowering the average recovery latency to 7.466 s. Fd-Stream rapidly reconstructs the state based on approximate backups and gradually compensates for recovery deviations through feedback regulation, further reducing the average recovery latency to 6.443 s. Compared with Full Backup, Fd-Stream reduces the recovery latency by 59.03%; compared with WAFP, it achieves a reduction of approximately 13.7%.

D. End-to-End Processing Latency

End-to-end processing latency reflects the total time consumed from when data enters the system at the input side to when processing is completed at the output side. It can characterize the impact of a fault tolerance mechanism on the main data path. The experiment runs for 60 minutes: the workload is set to 20,000 tuples/s during the first 30 minutes and is increased to 40,000 tuples/s during the last 30 minutes. Meanwhile, a Worker process is forcibly terminated every 10 minutes to simulate periodic failures. The system records the average end-to-end latency every 1 minute.

As shown in Fig. 7, during the stable low-load stage (0 - 30 min), the average end-to-end latency of Full Backup is approximately 125 ms, while WAFP reduces it to approximately 68 ms, and Fd-Stream further reduces it to approximately 42 ms. After the load increases to 40,000 tuples/s (from 31 min),

the steady-state latency of Full Backup rises to approximately 385 ms, whereas Fd-Stream can still keep the latency at approximately 65 ms. This is because Full Backup continuously performs full-state persistence, which significantly consumes computing and I/O resources under high-throughput scenarios. Since WAFP reduces part of the state persistence overhead, its latency is approximately 68 ms. In contrast, Fd-Stream reduces redundant state backups according to error feedback, thereby reducing interference with the main data path.

After a failure injection at every 10 minutes, Full Backup exhibits significant latency spikes due to full-state recovery and data replay, with peak latency exceeding 15,000 ms under high load. WAFP reduces the recovery data volume through approximate backup and therefore achieves lower peak latency. Fd-Stream reduces the data processing pressure during failure recovery through approximate state recovery, keeping the failure-induced latency peak at approximately 6,400 ms and returning to a stable latency level more quickly after failure. The experimental results show that, in complex stateful stream applications, Fd-Stream can effectively reduce both normal-operation latency and latency spikes under failure impact.

VI. Conclusion and Future Work

This paper proposes Fd-Stream, a feedback-driven approximate fault tolerance framework for stateful stream computing. Fd-Stream estimates global output error at low cost through anchor sampling, dynamically adjusts backup sampling rates according to operator importance, and rapidly restores operator states from approximate backups with subsequent feedback compensation. By integrating these techniques, Fd-Stream achieves a dynamic balance among output accuracy, fault tolerance overhead, and recovery latency. Compared with representative baseline methods, Fd-Stream significantly reduces resource utilization and recovery latency, demonstrating its failure recovery efficiency.

Our future work will mainly focus on extending the adaptability of Fd-Stream to complex stream processing scenarios and different data distributions, and integrating resource scheduling mechanisms to achieve a coordinated trade-off among accuracy, latency, and resource utilization.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 62372419; the Fundamental Research Funds for the Central Universities under Grant No. 265QZ2021001.

References

[1] M. Fragkoulis, P. Carbone, V. Kalavri, and A. Katsifodimos, "A survey on the evolution of stream processing systems," *The VLDB Journal*, vol. 33, no. 2, pp. 507–541, 2024.

[2] X. Liu and R. Buyya, "Resource management and scheduling in distributed stream processing systems: a taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–41, 2020.

[3] B. Del Monte, S. Zeuch, T. Rabl, and V. Markl, "Rethinking stateful stream processing with rdma," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 1078–1092.

[4] V. Cardellini, F. Lo Presti, M. Nardelli, and G. R. Russo, "Runtime adaptation of data stream processing systems: The state of the art," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–36, 2022.

[5] R. Gu, H. Yin, W. Zhong, C. Yuan, and Y. Huang, "Meces: Latency-efficient rescaling via prioritized state migration for stateful distributed stream processing systems," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 539–556.

[6] Y. Qing and W. Zheng, "Towards fine-grained scalability for stateful stream processing systems," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 2025, pp. 3835–3848.

[7] J. Zhao, H. Liu, S. Zhang, Z. Duan, X. Liao, H. Jin, and Y. Zhang, "Fast parallel recovery for transactional stream processing on multicores," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 1478–1491.

[8] Z. Xu, P. Liu, Q. Xia, W. Liang, G. Xu, W. Xu, P. Zhou, and H. Li, "Efficient and fault tolerant data stream processing with uncertain data rates in serverless edge computing," *IEEE Transactions on Services Computing*, 2025.

[9] S. Zhang, J. Soto, and V. Markl, "A survey on transactional stream processing," *The VLDB Journal*, vol. 33, no. 2, pp. 451–479, 2024.

[10] Z. Zhang, T. Liu, Y. Shu, S. Chen, and X. Liu, "Dynamic adaptive checkpoint mechanism for streaming applications based on reinforcement learning," in *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2023, pp. 538–545.

[11] H. Xu, P. Liu, S. T. Ahmed, D. Da Silva, and L. Hu, "Adaptive fragment-based parallel state recovery for stream processing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 8, pp. 2464–2478, 2023.

[12] Z. Cheng, Q. Huang, and P. P. Lee, "On the performance and convergence of distributed stream processing via approximate fault tolerance." *VLDB Journal International Journal on Very Large Data Bases*, vol. 28, no. 5, 2019.

[13] J. Francisco, M. E. Coimbra, P. F. R. Neto, F. Freitag, and L. Veiga, "Stateful adaptive streams with approximate computing and elastic scaling," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, 2023, pp. 174–183.

[14] S. Jayasekara, S. Karunasekera, and A. Harwood, "Optimizing checkpoint-based fault-tolerance in distributed stream processing systems: Theory to practice," *Software: Practice and Experience*, vol. 52, no. 1, pp. 296–315, 2022.

[15] X. Wang, C. Zhang, J. Fang, R. Zhang, W. Qian, and A. Zhou, "A comprehensive study on fault tolerance in stream processing systems," *Frontiers of Computer Science*, vol. 16, no. 2, p. 162603, 2022.

[16] G. Siachamis, K. Psarakis, M. Fragkoulis, A. Van Deursen, P. Carbone, and A. Katsifodimos, "Checkmate: Evaluating checkpointing protocols for streaming dataflows," in *2024 IEEE 40th international conference on data engineering (ICDE)*. IEEE, 2024, pp. 4030–4043.

[17] Y. Mao, Y. Huang, R. Tian, X. Wang, and R. T. Ma, "Trisk: Task-centric data stream reconfiguration," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 214–228.

[18] Y. Zhuang, Y. Gao, C. Guo, J. Wang, Y. Lin, and J. Sun, "Workload-aware approximate backup to reduce fault-tolerant overhead for stream processing applications," *Future Generation Computer Systems*, p. 108246, 2025.

[19] Takdir, H. Kitagawa, and T. Amagasa, "Local recovery and partial snapshot in distributed stateful stream processing: Takdir et al." *Knowledge and Information Systems*, vol. 67, no. 10, pp. 9407–9435, 2025.

[20] B. Liu, M. Ye, S. Wright, P. Stone, and Q. Liu, "Bome! bilevel optimization made easy: A simple first-order approach," *Advances in neural information processing systems*, vol. 35, pp. 17 248–17 262, 2022.

[21] Yahoo, "Yahoo streaming benchmarks," 2016, gitHub repository. Available: https://github.com/yahoo/streaming-benchmarks.

[22] Yahoo Webscope, "Yahoo webscope s5 dataset," 2015, available: https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70.

[23] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 147–156.